ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

EVALUACIÓN EXPERIMENTAL DE DETECTORES DE CARAS Y CARACTERÍSTICAS FACIALES

Raquel Solá Asenjo

Rafael Cabeza Laguna

Pamplona, 31 de enero de 2011

AGRADECIMIENTOS

Gracias a Rafa, mi tutor de proyecto, por darme la oportunidad de superarme a mí misma realizando este proyecto y por ayudarme a conocer una nueva rama en la que poder trabajar y desarrollar todo lo aprendido durante la carrera.

Gracias a mi familia, en especial a mis padres; por su ejemplo de superación, lucha y honestidad, por todo el apoyo mostrado, por la paciencia infinita, la comidica rica y por todo el cariño recibido, y a mi tato; por los viajes mañaneros en coche, “¿ta gusta’o?”.

Gracias a todos y a cada uno de mis “modelos” en la realización de la base de datos.

Gracias a mis amigos “telecos” porque sin ellos no sería lo mismo. Qué gran viaje a cuba, que grandes barbacoas, que grandes cenas, que grandes casas rurales… ¡¡Qué grandes momentos!!

Gracias a mis “corderas”, por su gran apoyo, necesario en momentos difíciles y por hacer divertidos hasta los días más aburridos.

Gracias a mis compañeros de master, por su preocupación y gran ayuda.

Gracias a Ainara, por enseñarme a confiar en mí misma, porque sin ella no sería lo que soy.

Y gracias a “Pepe”, por escucharme, por apoyarme, por animarme, por comprenderme, por ayudarme y sobre todo, gracias por hacerme siempre sonreír.

ÍNDICE

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

El concepto de eye-tracking hace referencia a un conjunto de tecnologías que permiten monitorizar y registrar la forma en la que una persona mira una determinada escena o imagen, en concreto en qué áreas fija su atención, durante cuánto tiempo y qué orden sigue en su exploración visual.

Las técnicas de eye-tracking tienen un gran potencial de aplicación en una amplia variedad de disciplinas y áreas de estudio, desde el marketing y la publicidad hasta la investigación médica o la psicolingüística, pasando por los estudios de usabilidad.

En el contexto de la Interacción Persona-Ordenador (IPO), podemos diferenciar dos posibles vertientes de aplicación del eye-tracking:

- Uso como dispositivo de entrada o interacción
- Uso como herramienta para la evaluación objetiva de interfaces

Aunque la precisión del eye-tracking como dispositivo de entrada dista de la de otros, como el ratón o el teclado, puede tener numerosas aplicaciones prácticas, tales como su uso en entornos de realidad virtual o por usuarios con discapacidad motriz.

Existe una gran variedad tecnológica de sistemas de eye-tracking, cada uno con sus propias ventajas e inconvenientes. Una de las técnicas más precisas implica el contacto físico con el ojo a través de un mecanismo basado en lentes de contacto, pero inevitablemente estos sistemas resultan muy incómodos para los usuarios. La mayoría de sistemas actuales son mucho menos molestos, ya que se basan en el uso de cámaras (eye-trackers) que proyectan rayos infrarrojos hacia los ojos del participante, sin necesidad de contacto físico. Los sistemas basados en eye-trackers requieren de un proceso previo de calibración, aunque esto no resulta un problema excesivamente grave al tratarse de un proceso bastante sencillo y rápido.

Dentro del campo de la IPO, en el caso de aplicación de eye-tracking como dispositivo de entrada o interacción, una posibilidad es el manejo del ordenador mediante los ojos. Este sistema, que supone una revolución para pacientes aquejados de paraplejias que impidan el movimiento, sustituye al ratón en las aplicaciones para Windows.

Este método permite al usuario colocar el puntero del ratón en cualquier lugar de la pantalla simplemente mirando a ese punto. Para hacer clic, el usuario no tiene más que parpadear lentamente o bien quedarse mirando fijamente en el punto durante un tiempo predeterminado. El sistema no discrimina a usuarios que emplean gafas y/o lentillas y funciona con independencia del color y tamaño de los ojos.

El principal problema de esta tecnología radica en el precio, ya que se mueve en torno a los 8000 euros. En un caso ideal este sistema se podría aplicar eliminando los infrarrojos, convirtiendo su precio en uno mucho más asequible. La intención es crear un sistema de eye-tracking para el control de ordenador que no necesitase los infrarrojos, es decir, que una cámara web sencilla fuese capaz de detectar el movimiento de los ojos. Para ello el primer paso es detectar la cara en una imagen captada por cámara web.

Dentro del marco general de la detección de objetos, la detección de caras ha centrado numerosos estudios e investigaciones, motivados tanto por su importancia a nivel neurobiológico, como por el enorme abanico de aplicaciones prácticas en las que sería utilizable (identificación biométrica, monitorización de individuos, compresión de vídeo,…).

La detección facial se encarga de determinar si existe o no alguna cara en una imagen dada y, en caso de encontrarse, extraer la localización y el contenido de dicha cara. En general este es un problema muy complejo ya que los objetos a detectar pueden ser de diversos colores, expresiones o poses y encontrarse en diferentes situaciones de iluminación.

La detección de caras lleva desarrollándose desde hace varias décadas. En los años 70 surgieron los primeros algoritmos, pero las investigaciones se abandonaron debido a la falta de utilidad por la falta de desarrollo de la técnica. En la actualidad esta línea de investigación tiene numerosas aplicaciones, siendo el ámbito comercial con las cámaras de fotos, el de la seguridad y la investigación criminal con el control a zonas restringidas y el control de fronteras los más destacados. Por otro lado, la mayoría de los algoritmos de detección facial pueden extenderse para el procesamiento de otros objetos como coches y peatones o para la localización de áreas faciales concretas.

Fruto de estos trabajos han surgido multitud de algoritmos y métodos para detectar caras en imágenes, entre los que a día de hoy, destaca el desarrollado por Paul Viola y Michael Jones [1]. Los extraordinarios resultados que han obtenido, unidos a la eficiencia y versatilidad de su esquema, hacen del detector de Viola-Jones un punto de partida muy interesante para cualquier aplicación donde se necesite detectar objetos con variabilidad estructural en tiempo real.

1.2 OBJETIVO Y ENFOQUE

1.2.1 Objetivo general

El objetivo fundamental del proyecto es la evaluación experimental y posterior optimización de un módulo previamente implementado para la detección de caras y características faciales (ojos, nariz y boca) orientado a su integración en un sistema de *eyetracking*. Este sistema hace referencia al proceso de calcular el punto donde se fija la mirada o el movimiento del ojo en relación con la cabeza.

Existen diversos sistemas para determinar el movimiento de los ojos, siendo la variante más común la que utiliza imágenes de vídeo a partir de la cuales se extrae la posición del ojo. En el caso de este estudio, las imágenes de video se obtendrán desde una webcam y serán evaluadas en tiempo real.

En el escenario propuesto, la manera más intuitiva de integrar la secuencia de vídeo procedente de la webcam en el módulo de detección sería aplicar el detector sobre cada frame, tratándolo como una única imagen.

Como en una secuencia de vídeo se encuentra una alta cantidad de fotogramas, se ha elaborado una base de datos de caras con más de 2000 imágenes tomadas en diferentes condiciones de iluminación y con diferente geometría de la cara. Estas imágenes, así como las etiquetas que determinan la posición de la cara y características faciales de cada una, serán posteriormente utilizadas para testear el módulo.

Debido a los diferentes parámetros que se pueden modificar a la hora de ejecutar el módulo de detección, se creará un algoritmo que compruebe el rendimiento del mismo utilizando diferentes valores para esos parámetros. Este algoritmo, que utilizará el módulo para procesar cada imagen, se encargará de comprobar si en cada procesado la detección fue exitosa o no comparando los resultados con las etiquetas reales de la base de datos.

1.2.2 Objetivos concretos

Construcción de una base de datos de caras con las siguientes características:

Variación de iluminación, posición, gestos, tamaños, distancia, presencia de componentes estructurales (gafas, pelo, barba…).
Múltiples resoluciones.
Color y en escalas de grises.
Anotación de caras y características faciales: rectángulo mínimo.
Unificación de otras bases de datos.

Definición de parámetros de test

TP, TN, FP, FN, FPR, TPR,…
Preparación de script para cálculo sobre la base de datos.

Versión opencv200

Interfaz con Matlab

Detección de caras.
Pruebas con diferentes xml, parámetros, opciones, escalas, ecualización, etc.
Detección de ojos sobre ROI
Pruebas con diferentes xml, parámetros, opciones, escalas, ecualización, etc.
Detección de boca sobre ROI
Pruebas con diferentes xml, parámetros, opciones, escalas, ecualización, etc.
Detección de nariz sobre ROI
Pruebas con diferentes xml, parámetros, opciones, escalas, ecualización, etc.

FDTOOL

Pruebas básicas. Resultados.
Documentación de entrenamiento.

2 MÉTODOS DE DETECCIÓN FACIAL

2.1 INTRODUCCIÓN

Las imágenes que contienen caras son esenciales en muchos sistemas inteligentes de visión y muchos esfuerzos en el procesado facial incluyen el reconocimiento y el seguimiento facial, la estimación de la pose y el reconocimiento de expresiones. Sin embargo, muchos métodos asumen que los rostros de una imagen se encuentran identificados y localizados. La detección de caras es una etapa fundamental en cualquier aplicación donde se realice algún tipo de análisis facial como por ejemplo reconocimiento de caras, codificación de vídeo en videoconferencias, interfaces inteligentes hombre-máquina, etc. El objetivo de esta etapa consiste en detectar y localizar la posición de un número indefinido de caras en una imagen sin importar su pose, orientación o condiciones de luz. En general, la detección de caras es un problema muy complejo ya que los objetos a detectar pueden ser de diferentes colores, expresiones, poses, tamaños relativos o tener condiciones de iluminación muy dispares.

- **Pose.** El ángulo de la cámara al capturar las imágenes puede variar rotando una cara hasta 180º desde una posición frontal dando lugar a poses frontales, de perfil, rotadas 45º, tomadas desde arriba o desde abajo… La orientación de la imagen también puede producir oclusiones en algunas características faciales como la nariz o los ojos.

Figura. 2.1. Imágenes de un sujeto en diferentes poses.

- **Presencia o ausencia de componentes estructurales.** La presencia de características faciales como barba, bigote y gafas pueden estar o no presentes e influyen en el rendimiento del sistema y presentan un alto grado de variabilidad.

Figura. 2.2. Imágenes de presencia de componentes estructurales. (a) Barba, (b) gafas y (c) bigote.

- **Expresión facial.** La apariencia de las caras está directamente afectada por la expresión facial de la persona.

Figura. 2.3. Imágenes de diferente expresión facial.

- **Oclusión.** Una cara puede verse totalmente o estar parcialmente tapada por diversos objetos o por la pose del usuario. En una imagen de un grupo de personas algunas caras pueden estar parcialmente tapadas por otras.

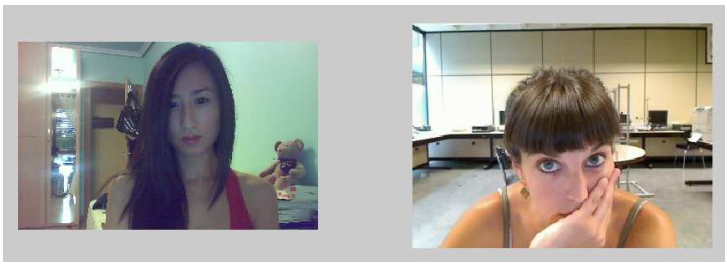


Figura. 2.4. Imágenes de oclusión facial. (a) Oclusión parcial del ojo. (b) oclusión de la boca.

- **Orientación de la imagen.** La rotación de una imagen afecta la posible localización de caras.

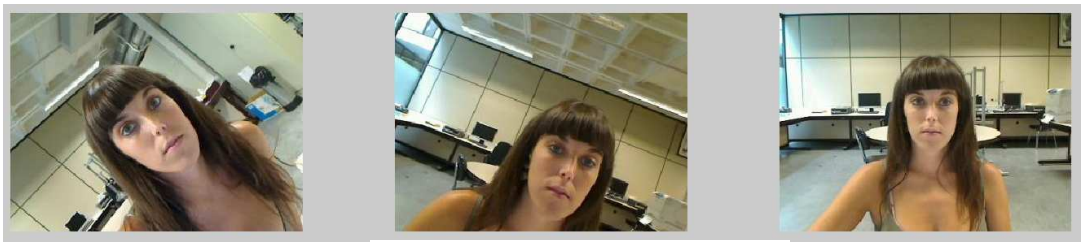


Figura. 2.5. Imágenes con distinta orientación.

- **Condiciones de captura.** Cuando una imagen es capturada, factores como la iluminación y las características de la cámara pueden afectar a la apariencia de una cara.

Figura. 2.6. Imágenes con diferentes condiciones de captura.

Debido a las variaciones comentadas anteriormente, se han propuesto numerosos métodos para la detección de caras, pero básicamente se podrían agrupar en cuatro grandes categorías:

- **Métodos basados en el conocimiento.** Estos métodos se basan en normas que codifican el conocimiento humano de cómo se constituye una cara típica. Por lo general, las normas utilizan la relación entre los rasgos faciales. Estos métodos son diseñados principalmente para la localización de la cara.

- **Métodos de aproximaciones a característica invariantes.** Estos algoritmos tienen el objetivo de encontrar las características estructurales que existen incluso cuando la pose, el punto de vista, o las condiciones de iluminación varían. Estos métodos son diseñados principalmente para la localización facial.

- **Métodos de coincidencia de plantillas.** Varios patrones estándar de una cara se almacenan para describir la cara en su conjunto o los rasgos faciales por separado. Las correlaciones entre una imagen de entrada y los patrones almacenados se utilizan para realizar la detección. Estos métodos se utilizan tanto para la localización como para la detección de la cara.

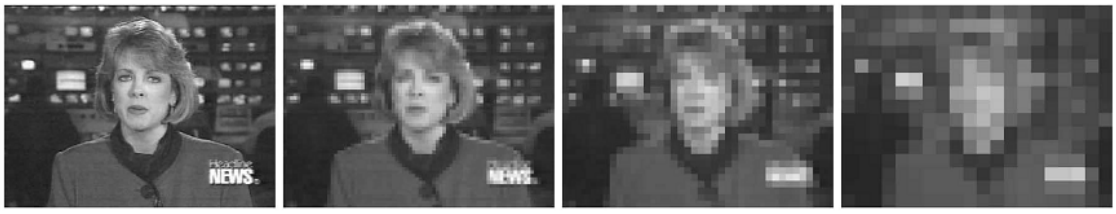
- **Métodos basados en la apariencia.** En contraste con los métodos de coincidencia, los modelos (o plantillas) son obtenidos de un conjunto de imágenes de entrenamiento en los que se capta la variabilidad representativa de la apariencia facial. Estos modelos aprendidos se utilizan para la detección. Estos métodos están diseñados principalmente para la detección de rostros.

2.2 MÉTODOS BASADOS EN EL CONOCIMIENTO

Los métodos de detección facial basados en el conocimiento se fundamentan en una serie de reglas definidas antes de la implementación del sistema que codifican el conocimiento sobre lo que constituye una cara. Generalmente estas reglas capturan las relaciones entre rasgos faciales. Es fácil llegar a reglas simples para describir las características de una cara. Por ejemplo, una cara aparece con frecuencia en una imagen con dos ojos que son simétricos entre sí, una nariz y una boca. Las relaciones entre las características pueden representarse por las distancias relativas entre ellas y su posición. Las características faciales en una imagen de entrada se extraen en primer lugar, y las posibles caras son identificadas basándose en las reglas codificadas. Por lo general, un proceso de verificación es aplicado para reducir falsas detecciones.

Este tipo de métodos poseen varias desventajas como la dificultad para la traducción del conocimiento humano en reglas bien definidas. Si se detallan las normas en exceso (reglas estrictas), pueden no detectarse caras que no pasan todas las reglas, dan lugar a una baja tasa de detección. Si las normas son demasiado generales, pueden darse muchos falsos positivos. Otra desventaja es la dificultad de ampliar estos métodos para detectar rostros en diferentes poses ya que sería demasiado extenso crear reglas para todos los casos posibles.

Yang y Huang utilizaron un método basado en el conocimiento jerárquico para detectar las caras [9]. Su sistema consta de tres niveles de normas. En el nivel más alto de la jerarquía, se buscan las posibles regiones que contenga una cara aplicando reglas generales de cómo es una cara, esto se realiza en la imagen de entrada disminuyendo su resolución, estas posibles regiones son enviadas hacia el siguiente nivel en el que se ecualiza la imagen y se realiza una detección de bordes. Las normas del nivel superior son descripciones generales de lo que es una cara, mientras que las normas en los niveles inferiores son más estrictas, dependen de los detalles de los rasgos faciales. Se crea una jerarquía multi-resolución de imágenes por promediado y submuestreo, en la figura. 2.7 se muestra un ejemplo.



(a) (b) (c) (d)

Figura. 2.7. (a) n = 1, imagen original. (b) n = 4. (c) n = 8. (d) n = 16. Original y las correspondientes imágenes de baja resolución. Cada celda se compone de *n* x *n* píxeles en los que se sustituye la intensidad de cada píxel por la intensidad media de los píxeles en esa celda.

Ejemplos de las reglas utilizadas para localizar candidatos a caras en imágenes con resolución más baja son: la parte central de la cara (la parte sombreada más oscura de la figura. 2.8) tiene cuatro celdas con una intensidad básicamente uniforme , la parte más externa que rodea la anterior (las partes ligeramente sombreada en la figura. 2.8) tiene también básicamente una intensidad uniforme , y la diferencia entre la media de los valores de gris de la parte central y la parte que la rodea es significativa. La resolución más baja (nivel 1) de la imagen es buscada por las posibles caras y se procesan con resoluciones más finas. En el nivel 2, se realiza una ecualización local del histograma en las posibles caras, seguida por la detección de bordes. Los candidatos a caras que han sobrevivido al nivel 1 y 2 se examinan en el nivel 3 con otro conjunto de reglas que responden a características faciales como los ojos y la boca.

Figura. 2.8. Cara típica utilizada en métodos basados en el conocimiento.

A pesar de que este método no da lugar a una alta tasa de detección, la idea de la utilización de una jerarquía multi-resolución y el uso de normas para orientar las búsquedas de caras se han utilizado en la detección de caras en trabajos posteriores.

Kotropoulos y Pitas [11 y 12], presentaron un método de localización basado en normas que es similar al visto anteriormente [9]. En primer lugar, los rasgos faciales son localizados con un método de proyección que Kanade utilizó con éxito para encontrar el contorno de una cara [13].

Sea $I = (x, y)$ el valor de la intensidad de una imagen mxn en la posición (x, y), las proyecciones horizontal y vertical de la imagen se definen como:

$$HI(x) = \sum_{y=1}^{n} I(x, y) \quad VI(y) = \sum_{x=1}^{m} I(x, y) .$$

El perfil horizontal de una imagen de entrada se obtiene en primer lugar, y luego los dos mínimos locales determinados mediante la detección de cambios bruscos en HI,

éstos mínimos corresponden al lado izquierdo y derecho de la cabeza. El perfil vertical es obtenido del mismo modo, y los mínimos locales determinan las ubicaciones de los labios de la boca, la punta de la nariz y los ojos. Estas características detectadas constituyen un candidato facial. La figura.2.9 a) muestra un ejemplo en el cual los contornos de la cara corresponden al mínimo local, donde se dan los cambios más abruptos de intensidad. Posteriormente, las normas de detección de las cejas, ojos, fosas nasales, nariz, y boca se utilizan para validar estos candidatos.

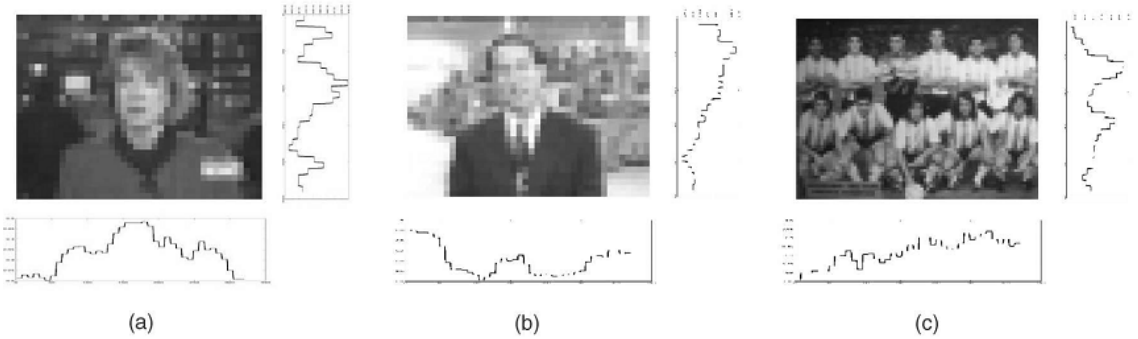


Figura. 2.9. (a) y (b) n = 8. (c) n = 4. Perfiles horizontal y vertical.

El método de proyección puede ser eficaz para detectar una sola cara en un fondo uniforme. Sin embargo, el mismo método tiene dificultades para detectar caras en fondos complejos o múltiples caras.

Otro método basado en el conocimiento es el propuesto por Ming-Hsuan Yang y Narendra Ahuja [14].

Proponen un algoritmo de localización de la cara basado en el color de la piel que puede encontrar una cara en un entorno complejo que incluye regiones que no son la cara pero con el mismo color de la piel, por ejemplo, las manos.

El algoritmo se divide en tres pasos. El primer paso implica el uso de componentes de color para segmentar el color de la piel del fondo complejo, esto se lleva a cabo a partir de una distribución bidimensional Gaussiana. El segundo paso detecta una curva llamada curva de división que delimita la frontera entre la región del pelo y la región de la cara. En el último paso, la curva de división se utiliza para estimar la región de la cara.

Este método puede detectar caras en imágenes en color independientemente de su tamaño, orientación y punto de vista de manera bastante efectiva, pero tiene una gran limitación, el individuo debe tener pelo.

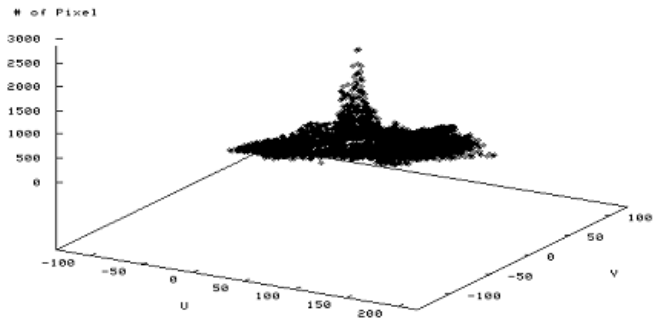


Figura. 2.10. Distribución de color de la piel humana

2.3 MÉTODOS DE APROXIMACIONES A CARACTERÍSTICAS INVARIANTES

Estos métodos se basan en el hecho de que los seres humanos sin esfuerzo pueden reconocer rostros u objetos en diferentes poses o condiciones de iluminación por lo que deben existir propiedades o rasgos en los rostros que son invariables a pesar de la variación de ciertas condiciones en la imagen. A partir de este supuesto numerosos métodos han sido propuestos para detectar primero los rasgos faciales y luego deducir la presencia o no de una cara. Los rasgos que se tratan de localizar son principalmente los ojos, cejas, nariz, boca y ciertas relaciones entre ellos y se suelen extraer utilizando detectores de bordes. Basándose en las características extraídas, se construye un modelo estadístico para describir sus relaciones y verificar la existencia de una cara.

El principal problema de estos métodos es la dificultad de localizar ciertos rasgos faciales debido a la iluminación, oclusión, presencia de ruido en la imagen.

Figura. 2.11. Varias características faciales

2.3.1 Características faciales

Sirohey propuso un método de localización que separa la cara de un fondo complejo para su identificación [15]. Utiliza un mapa de bordes (detector de Canny) que los elimina y agrupa para que sólo los bordes que están en el contorno de la cara se conserven. Se coloca una elipse para marcar el límite entre la región de la cabeza y el fondo.

Figura. 2.12. (a) Intensidad de la imagen, (b) mapa de bordes del detector de Canny y (c) elipse.

En lugar de utilizar los bordes, Chetverikov y Lerch presentaron un método simple de detección de rostros usando manchas y rayas (secuencias lineales de orientación similar a los bordes) [16]. Su modelo de la cara se compone de dos regiones oscuras y tres claras para representar los ojos, los pómulos y la nariz. El modelo se vale de

rayas para representar los contornos de la cara, las cejas y los labios. Dos configuraciones triangulares se utilizan para codificar la relación espacial entre las representaciones de las rayas. Se aplica un Laplaciano a la imagen de baja resolución para facilitar la detección de las regiones. A continuación, la imagen se escanea para encontrar candidatos a triángulos. Una cara se detecta si las líneas están alrededor de los triángulos candidatos.

K.C. Yow y R.Cipolla [18] plantearon un marco de detección de rostros que agrupa las características de la imagen en entidades reconocibles utilizando la organización perceptiva, asignando probabilidades a cada una de ellas, y reforzando estas probabilidades con técnicas de razonamiento bayesiano.

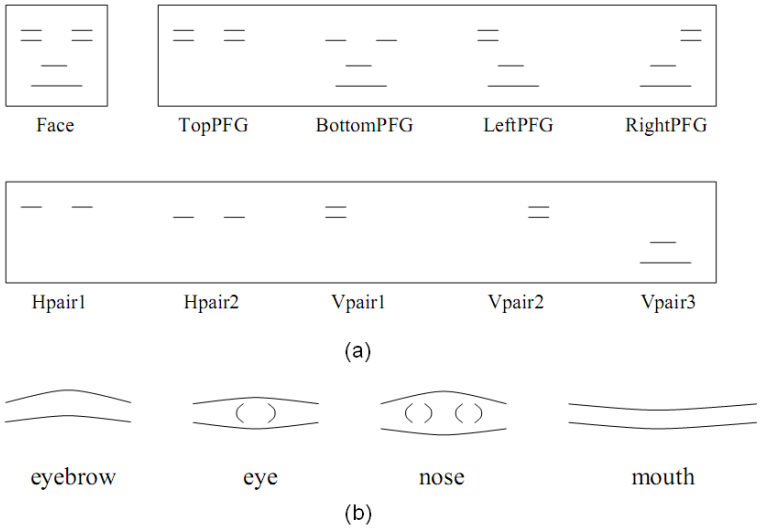


Figura. 2.13. (a) modelo de la cara y sus componentes y (b) modelos de rasgos faciales.

Y. Amit, D. Geman, y B. Jedynak emplearon un método para la detección de formas y la aplicaron para detectar caras frontales en imágenes [19]. La detección consta de las siguientes dos etapas: enfoque y clasificación intensiva. El enfoque se basa en las disposiciones espaciales de fragmentos del borde extraídos de un simple detector de borde mediante diferencia de intensidad.

2.3.2 Textura

Los rostros humanos tienen una textura distinta que puede ser usada para separarlos de diferentes objetos. Augusteijn y Skufca desarrollaron un método que infiere la presencia de una cara mediante la identificación de texturas como el rostro [20]. La textura se calcula utilizando características estadísticas de segundo orden (SGLD) [21] en subimágenes de 16 x 16 píxeles. Se consideran tres tipos de características: piel, pelo, y otros. Utilizaron una correlación en cascada de redes neuronales para la clasificación supervisada de texturas y un mapa de Kohonen de auto-organización de características para formar grupos de diferente textura. Para inferir la presencia de una cara en los niveles de la textura, sugieren que se empleen la cantidad de veces que aparecen el pelo y las texturas de la piel.

Dai y Nakano también aplicaron el modelo SGLD para hacer la detección de caras [22]. La información del color se incorpora al modelo de textura. Utilizando el modelo de textura, diseñan un plan de exploración para la detección de rostros de color en las escenas en las que las partes anaranjadas, incluidas las zonas de la cara, son

mejoradas. Una de las ventajas de este método es que puede detectar caras que no están en posición vertical o tienen características tales como barba y gafas.

2.3.3 Color de la piel

El color de piel humana se ha utilizado y ha demostrado ser una función eficaz en muchas aplicaciones de detección de rostros por seguimiento. A pesar de que diferentes personas tienen diferente color de piel, varios estudios han demostrado que gran diferencia radica en la intensidad en lugar de en su crominancia. Varios espacios de color han sido utilizados para etiquetar píxeles como piel incluyendo RGB, RGB normalizado, HSV (o HSI), YCrCb, YIQ, YES, CIE XYZ y CIE LUV.

Muchos métodos han sido propuestos para construir un modelo de color de la piel. El modelo más simple es definir una región de tono de la piel usando los valores Cr, Cb, es decir, *R(Cr,Cb)*, a partir de muestras de píxeles del color de la piel. Con umbrales elegidos cuidadosamente, $[Cr_1, Cr_2]$ y $[Cb_1, Cb_2]$ un píxel se clasifica como tono de piel si los valores *(Cr,Cb)* pertenecen a de los intervalos, es decir, $Cr_1 \leq Cr \leq Cr_2$ y $Cb_1 \leq Cb \leq Cb_2$. Crowley y Coutaz utilizan un histograma *h(r,g)* de valores *(r,g)* en el espacio de color normalizado RGB para obtener la probabilidad de obtener un determinado vector RGB teniendo en cuenta que el pixel cumple con el color de la piel. En otras palabras, un píxel es clasificado como perteneciente al color de la piel si $h(r,g) \geq \tau$, dónde τ es un umbral establecido empíricamente a partir de las muestras del histograma.

Saxe y Foulds [23] propusieron un método iterativo de identificación de la piel que utiliza la intersección del histograma en el espacio de color HSV. Un ajuste inicial de píxeles del color de la piel, llamado control de semillas, se elige por el usuario y se utiliza para iniciar el algoritmo iterativo. Para detectar las regiones de color de la piel, su método mueve a través de la imagen el control de semillas, y presenta el histograma de control y el histograma actual de la imagen para la comparación. La intersección de los histogramas se utiliza para comparar el histograma de control y el actual. Si el número de casos en común, es decir, la intersección es mayor que un umbral, el actual campo de semillas se clasifica como color de la piel.

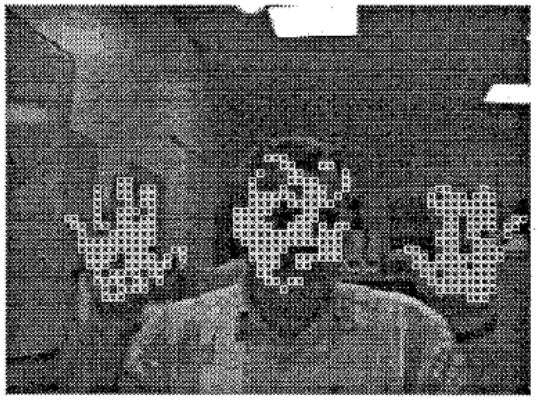


Figura. 2.14. Los resultados de ejecutar el algoritmo iterativo. El punto de control inicial fue elegido en la frente del sujeto.

Kjeldsen y Kender definen un color basado en el espacio de color HSV para separar la zona de la piel del fondo [24]. En contraste con los métodos no paramétricos mencionados anteriormente, funciones de densidad gaussiana y una mezcla de gaussianas se suelen usar como modelo de color de la piel. Los parámetros de una

distribución unimodal Gaussiana a menudo son estimados usando Máxima verosimilitud. La motivación para el uso de una mezcla de gaussianas se basa en la observación de que el histograma del color de la piel de las personas de diferentes orígenes étnicos no forma una distribución unimodal, sino más bien una distribución multimodal. Los parámetros en una mezcla de gaussianas suelen ser estimados mediante un algoritmo EM. Recientemente, Jones y Rehg llevaron a cabo un experimento a gran escala en el que se recogen cerca de mil millones de tonos de piel etiquetados (en el espacio de color RGB normalizado).

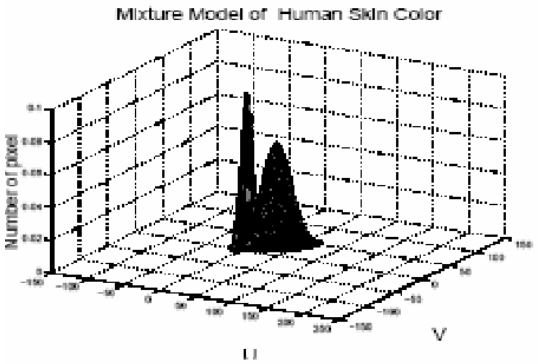


Figura. 2.15. Función de densidad estimada.

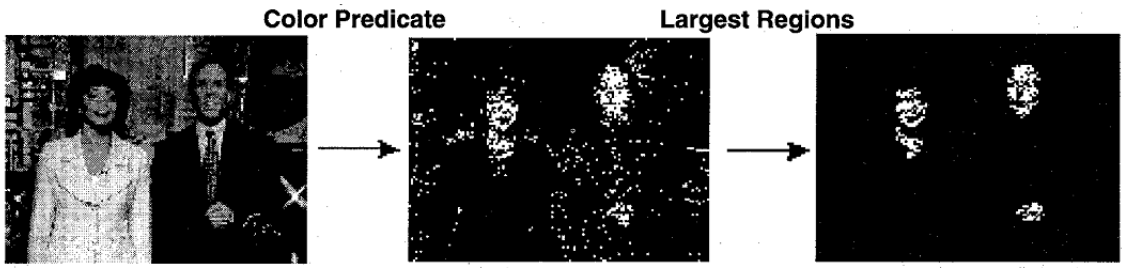


Figura. 2.16. Ejemplos de segmentación.

La información del color es una herramienta eficiente para la identificación facial y la identificación de rasgos faciales específicos, si el modelo de color de la piel es convenientemente adaptado para los diferentes ambientes de iluminación. Sin embargo, estos modelos de color de la piel no son eficaces cuando el espectro de la fuente de luz varía de forma significativa. En otras palabras, la apariencia del color es a menudo inestable debido a los cambios de iluminación tanto del fondo como del primer plano.

Aunque el problema del color se ha abordado mediante la formulación de modelos basados en la física, varias propuestas han sido planteadas para utilizar el color de la piel en diferentes condiciones de iluminación. McKenna, Raja y Gong presentan un modelo de color adaptativo para realizar un seguimiento de las caras en distintas condiciones de iluminación [25]. En lugar de confiar en un modelo de color de la piel basado en un color constante, utiliza un modelo estocástico para estimar distribución de color de un objeto y adaptarse a los cambios en los puntos de vista y las condiciones de iluminación. Los resultados preliminares muestran que su sistema puede detectar caras dentro de un rango de condiciones de iluminación. Sin embargo, este método no puede aplicarse para detectar caras en una única imagen.

Figura. 2.17. Fotogramas de una secuencia. Hay iluminación exterior direccional fuerte.

Por lo general, el color de la piel no es suficiente por sí solo para detectar o seguir las caras.

2.3.4 Múltiples características

Recientemente, numerosos métodos que combinan varias características faciales han sido propuestos para localizar o detectar caras. La mayoría de ellos utilizan las características globales como el color de la piel, el tamaño y la forma para encontrar posibles caras, y luego verifica estos candidatos utilizando características locales, tales como cejas, nariz, y pelo. Un modelo típico comienza con la detección de regiones de la piel como se describe en la sección *2.2.3 Color de la piel*. A continuación, los píxeles tomados como piel, se agrupan utilizando el análisis de componentes conectadas o algoritmos de agrupamiento. Si la forma de una región conectada tiene forma elíptica u ovalada, se convierte en un candidato a cara. Por último, las características locales se utilizan para verificación.

Yachida, Chen y Wu presentaron un método para detectar caras en imágenes de color utilizando la teoría difusa [26]. Se utilizaron dos modelos difusos para describir la distribución de la piel y el color del pelo en el espacio de color CIE XYZ. Cinco modelos (una frontal y cuatro puntos de vista laterales) con forma de la cabeza se utilizan para extraer la presencia de caras en imágenes. Cada modelo de forma es un patrón en 2D que consiste en *m x n* celdas cuadradas donde cada celda puede contener varios píxeles. Dos propiedades son asignadas a cada celda: la proporción de piel y la proporción de pelo, que indican la proporción de la zona de piel (o la zona de pelo) dentro del área de la celda. En una imagen de prueba, cada píxel es clasificado como el pelo, la cara, el pelo o la cara, y el pelo o el fondo basándose en modelos de distribución, generando regiones como la piel y el pelo. Los modelos de forma de la cabeza se comparan con las zonas extraídas de la piel y del pelo en la imagen de prueba. Si son similares, la región se convierte en un candidato a cara. Para verificación, las características ojo-ceja y nariz-boca son extraídas de un candidato a cara utilizando bordes horizontales.

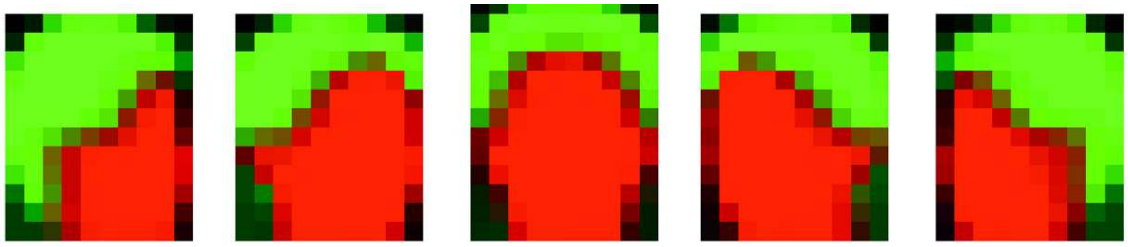


Figura. 2.18. Modelos primitivos de la forma de la cabeza

Sobottka y Pitas propusieron un método para la localización de la cara y la extracción de características faciales utilizando la forma y el color [27]. En primer lugar, la segmentación de color en el espacio HSV se realiza para localizar regiones como la piel. Se determinan componentes conectadas por regiones crecientes a una baja resolución. Para cada componente conectada, la elipse que mejor encaja se calcula utilizando momentos geométricos. Las componentes conectadas que están bien aproximadas por una elipse son seleccionadas como candidatas a cara. Posteriormente, estos candidatos son verificados por la búsqueda de rasgos faciales en el interior de las componentes conectadas. Características, tales como los ojos y la boca, se extraen apoyándose en la observación de que son más oscuras que el resto de la cara.

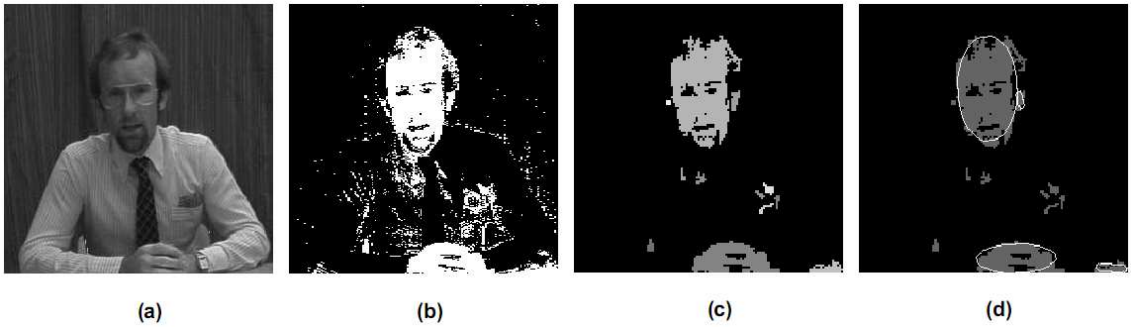


Figura. 2.19. (a) Imagen original, (b) Distribución de la piel, (c) Componentes conectadas, (d) Elipses de mejor ajuste.

La simetría de los patrones de la cara también se ha aplicado a la localización de la cara [28]. La clasificación de piel / no piel se lleva a cabo utilizando la función de densidad de clase condicional en el espacio de color YES seguido de un suavizado a fin de producir regiones contiguas. A continuación, una plantilla elíptica se utiliza para determinar la similitud de las regiones de color de la piel basándose en la distancia Hausdorff. Por último, los centros de los ojos son localizados usando varias funciones de coste que están diseñadas para aprovechar las simetrías inherentes asociadas a la cara y a las ubicaciones de los ojos. La punta de la nariz y el centro de la boca se encuentran utilizando la distancia entre los centros de los ojos. Una desventaja es que es eficaz sólo para una única cara frontal y cuando los dos ojos son visibles.

S.-H. Kim, N.-K. Kim, S.C. Ahn y H.-G. Kim emplearon la gama y el color para la detección de caras [29]. Se calculan mapas de disparidad y los objetos se separan del fondo con un histograma de disparidad utilizando la hipótesis de que los pixeles del fondo tienen la misma profundidad y que superan en número a los píxeles del primer plano. Con una distribución de Gauss normalizada en el espacio de color RGB, las regiones separadas y con un color como el de la piel se clasifican como caras.

2.4 MÉTODOS DE COINCIDENCIA DE PLANTILLAS

En los métodos de coincidencia de plantillas un patrón facial estándar (normalmente de una cara en pose frontal) se predefine manualmente o con parámetros mediante una función. Dada una imagen de entrada, se calculan unos valores de correlación utilizando los patrones estándar para el contorno facial, los ojos, la nariz y la boca independientemente. La existencia de una cara se determina en base a los valores de correlación. Estos métodos tienen como ventaja una implementación sencilla, sin embargo se ha probado que son ineficaces para la tarea de la detección facial ya que no pueden tratar efectivamente la variación en escala, pose y forma de la cara.

La aplicación de una plantilla es similar a los métodos basados en conocimiento, ya que el conocimiento de las características faciales o bien puede ser aprendido y construir así una plantilla dinámica, o bien puede ser predefinido. La aplicación es una plantilla podría realizarse a diferentes escalas debido a la simpleza de muchas plantillas que se relacionan simplemente con la simetría de las características faciales y con las distancias relativas. Sin embargo, como se ha dicho, utilizar solamente estos métodos no es muy efectivo con lo que a menudo suelen utilizarse junto a otras técnicas basadas en características invariantes. Los métodos descritos a continuación se centran sólo en aquellas técnicas donde la coincidencia de plantillas es la base para la detección.

2.4.1 Plantillas predefinidas

Sabert y Tekalp [30] utilizaron una función de coste basada en la simetría y un módulo de clasificación de formas basado en la deducción sobre un conjunto de muestra. Mediante el uso de un proceso de segmentación de piel esta técnica es capaz de trabajar basándose en una plantilla predefinida, que es usada junto a un modelo elíptico y un algoritmo de búsqueda de características faciales.

En la primera etapa se realiza una segmentación del color de la imagen. Esto no es muy diferente a las técnicas que usan características invariantes basadas en el color presentadas anteriormente. Sin embargo, hay una diferencia y es la aplicación del espacio YCbCr en vez del RGB, que es menos sensible a las variaciones de la intensidad. A continuación se realiza una clasificación del color de la piel, que se deriva de una distribución obtenida a partir de un conjunto de pruebas. Para agrupar las zonas de piel se utiliza un filtrado GRF (Gibbs Random Field). El uso del color como clasificador es muy eficiente computacionalmente y se usa como un proceso preliminar a la coincidencia de plantillas.

Una vez identificadas las regiones de piel debemos decidir si contienen una cara o no, esto se realiza principalmente mediante el uso de un modelo de clasificación de forma. El modelo es una plantilla de cara basada en la naturaleza elíptica de la cara humana. El centro de la elipse se determina mediante el cálculo de auto-vectores (vectores propios) que se derivan de las coordenadas espaciales de los pixeles de piel identificados en una región continua y se calcula su tamaño basándose en la distancia mínima de Hausdorff calculada a partir de los píxeles de la zona de piel agrupados alrededor del centro de la elipse. La plantilla se representa como una elipse y una función de coste de simetrías para las regiones de los ojos, la nariz y la boca.

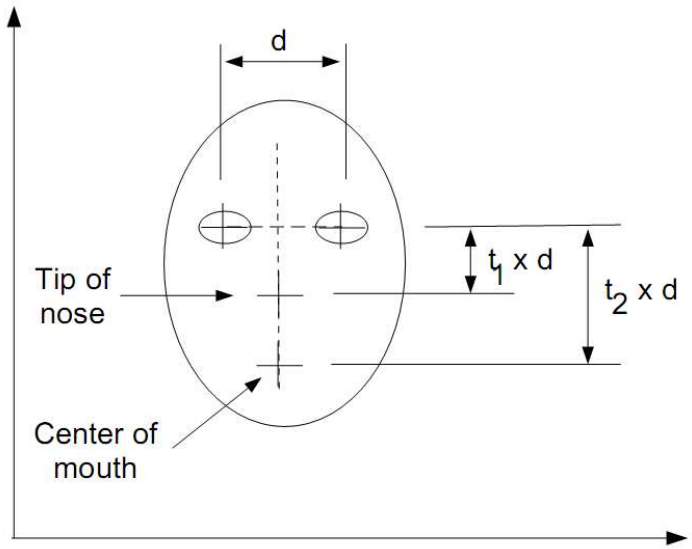


Figura. 2.20. Ubicación de la punta de la nariz y el centro de la boca

La función se basa en:

i. Los ojos están situados en una línea paralela al eje menor de la elipse.

ii. Los ojos son simétricos con respecto al eje mayor de la elipse, representado mediante la dirección del vector propio asociado con el mayor vector propio.

iii. Los ojos son equidistantes respecto al eje menor representado por la dirección del vector propio asociado al vector propio más pequeño.

iv. Los ojos son generalmente los huecos más próximos en la segmentación de la piel.

v. Los ojos están colocados sobre el eje menor de la elipse.

Utilizando la función de coste se realiza la decisión acerca de la existencia de una cara en las zonas de piel identificadas en la primera etapa del procedimiento. La aplicación de la segmentación del color es el paso clave sin el cual el modelo elíptico de la cara no podría formarse ni se podrían observar las características localizadas. Por lo tanto resulta clara la importancia de que exista algún tipo de mecanismo que identifique dónde debería colocarse la plantilla. En este caso se hace mediante la segmentación del color de piel.

P. Sinha emplea un pequeño conjunto de características espaciales invariantes en una imagen para describir el espacio de los patrones faciales [31]. Para designar lo invariante se basa en que, mientras que las variaciones en iluminación cambian el brillo individual de diferentes partes faciales (ojos, mejillas, frente) el brillo relativo de estas partes permanece aproximadamente invariable. Determinar estas proporciones en unas cuantas regiones y guardar solamente su dirección (por ejemplo, ¿Es una región más brillante que otra?) proporciona una característica invariante bastante robusta. Por lo tanto, las regularidades en el brillo facial se codifican como una plantilla de relaciones que es una plantilla espacial de una cara con subregiones que se pueden identificar con características faciales clave como ojos, mejillas y frente. Las relaciones en el brillo entre distintas zonas faciales se capturan mediante relaciones entre subregiones. Se dice que se ha localizado una cara si una imagen satisface todas las relaciones establecidas anteriormente.

La idea de utilizar diferencias de intensidad entre regiones adyacentes ha sido ampliada posteriormente a representaciones basadas en oscilaciones de ondas para peatones, coches y detección facial.

2.4.2 Plantillas deformables

A. Yuille, P. Hallinan y D. Cohen proponen una técnica que detecta y describe características faciales utilizando plantillas deformables que modelen esas características y que encajen en un modelo elástico a priori (por ejemplo los ojos) [32]. En esta técnica las características faciales se describen mediante plantillas, que han sido especificadas utilizando parámetros que posibilitan el conocimiento previo sobre la forma esperada de las características para guiar el proceso de detección. Estas plantillas son lo suficientemente flexibles como para poder cambiar su tamaño, y otros parámetros, para ajustarse a los datos. Se define una función de energía para unir bordes, picos y valles en la imagen de entrada con los correspondientes parámetros en la plantilla. El mejor ajuste del modelo elástico a la imagen se realiza minimizando la función de energía de los parámetros. Una vez hecho esto los parámetros de la plantilla se actualizan mediante descenso por gradiente. Cambiar estos parámetros se corresponde con una alteración en la posición, orientación, tamaño y otras propiedades de la plantilla. Aunque sus resultados experimentales hayan demostrado un buen rendimiento en el rastreo de características no rígidas, una desventaja de este método es que la plantilla debe ser inicializada en las cercanías del objeto de interés.

2.5 MÉTODOS BASADOS EN LA APARIENCIA

Estos algoritmos utilizan modelos o plantillas que capturan la variabilidad en el aspecto de los rostros humanos. En contraste con los métodos anteriores estos modelos son recogidos o aprendidos en base a imágenes de entrenamiento. En general este tipo de métodos se basan en técnicas de análisis estadístico y en aprendizaje automático para encontrar las características relevantes de imágenes de rostros e imágenes que no poseen rostros. Las características aprendidas toman la forma de modelos de distribución o funciones discriminantes que son usadas para la detección. Los algoritmos de este tipo han demostrado muy buenos resultados empíricos, suelen ser rápidos y robustos y pueden detectar rostros sin importar su pose y orientación. La desventaja es que se tiene una etapa de entrenamiento que necesita gran cantidad de ejemplos positivos y negativos para luego obtener buenos resultados.

2.5.1 Redes Neuronales

H. Rowley, S. Baluja, y T. Kanade [33] aplican un método basado en redes neuronales para proporcionar medios efectivos para la decisión. Se presenta como una red neuronal que se entrena para reconocer caras frontales. También combina algunas de estas redes para mejorar el rendimiento total del sistema. Los datos usados para entrenar cada red varían, específicamente con la frecuencia de imágenes que no contienen caras.

Cada red se entrena con imágenes con una resolución de 20 por 20 píxeles, donde cada ventana es pre-procesada para ecualizar la intensidad. La intensidad se normaliza con una función que también tiene en cuenta un modelo elíptico simple, el cual oculta los bordes de la ventana de entrenamiento en un intento de ocultar píxeles que potencialmente puedan pertenecer al fondo de la imagen. Las redes neuronales

tienen conexiones de retina del mismo modo que sus capas de entrada individuales, y un número real de valor único como salida que define la presencia de una cara.

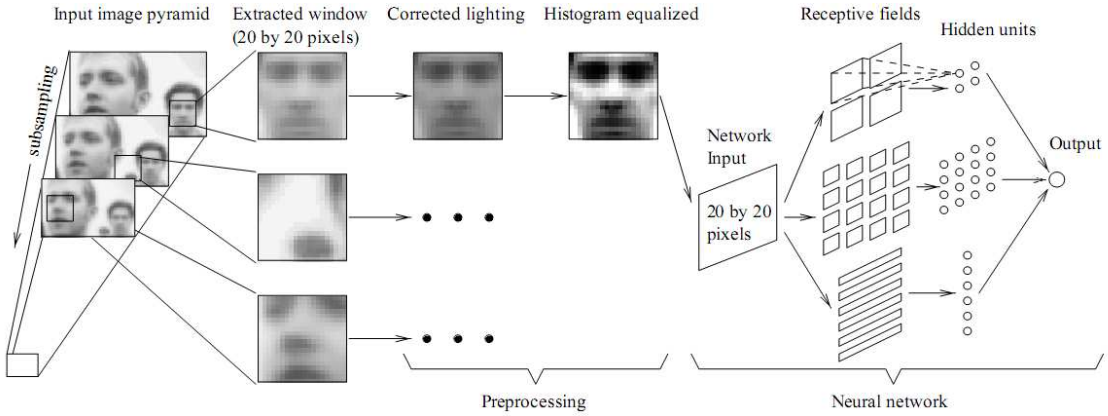


Figura. 2.21. Algoritmo básico para la detección facial

Uno de los beneficios más importantes de este método es su habilidad para utilizar múltiples redes entrenadas de manera diferente y por lo tanto proporcionar una decisión más exacta a la hora de localizar caras. Esto sólo es posible gracias a la naturaleza del método utilizado, simplemente entrenando redes individuales con pequeñas diferencias en los datos de entrenamiento obtenemos resultados completamente diferentes.

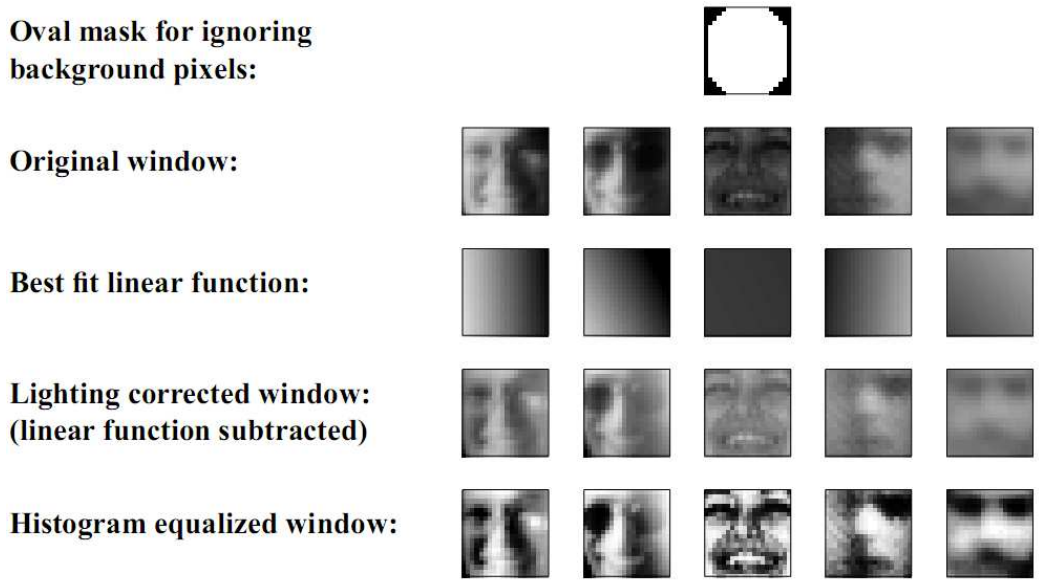


Figura. 2.22. Pasos de pre-procesamiento de una ventana.

2.5.2 Sparse Network of Winnows (SNOW)

M.-H. Yang, D. Roth, and N. Ahuja [34] proponen una técnica diferente denominada SNOW (Sparse Network of Winnows) la cual utiliza una red formada por unidades lineales para definir el espacio de las características aprendidas. Al utilizar su método se da una relevancia particular a las tareas de clasificación que tengan grandes conjuntos de características. Dicho método muestra buenos resultados en la fase de aprendizaje en dichas circunstancias. Mediante el uso de características booleanas primitivas la técnica codifica tanto la intensidad, como los datos de la posición de los píxeles en la imagen de muestra.

Los nodos en la capa de entrada de la red representan relaciones simples entre las entradas y se utilizan como características de entrada. Cada unidad lineal se llama nodo objetivo y representa las relaciones de interés entre los ejemplos de entrada. La decisión en este sistema se hace mediante la unión de los nodos objetivo, en este caso sólo dos: cara y no cara. Dado un conjunto de relaciones, es decir tipos de características, que puedan existir en una imagen, cada imagen se asocia a uno de esos conjuntos de características presentes y SNOW propagará estas relaciones a un nodo objetivo y se tomará una decisión.

La aplicación de SNOW da lugar a detectar caras en múltiples poses, expresiones y diferentes niveles de iluminación. La aplicación del algoritmo Winnow en la red proporciona una manera de aprendizaje que se ajusta a dominios donde el espacio de características es grande y desconocido. El uso de esta técnica se ha probado como una de las mejores implementaciones. Esto puede deberse a la aplicación del algoritmo SNOW y la red de decisión que se produce. La decisión es bastante robusta debido al uso del algoritmo Winnow.

2.5.3 Clasificadores Bayesianos

T. V. Pham, M. Worring, A. W. M. Smeulders presentan una red Bayesiana con una estructura en forma de bosque para resolver un problema de clasificación de una sola clase [35]. Se eligen los clasificadores Bayesianos debido a la gran velocidad que estos pueden alcanzar. La técnica genera un clasificador agregado ya que proporciona una manera natural de resolver los problemas de clasificación referidos a una sola clase.

El método de detección facial aquí referido se construye sobre una base de resolución de 20 pixeles que se escala con un factor de 1.2 hasta que el tamaño de la imagen sea menor que el de la ventana escalada. La resolución de 20 pixeles fue elegida al considerarse por parte de los autores más que adecuada para obtener las características de una cara humana y minimizar la relación entre la resolución y el tiempo de clasificación. En el conjunto de datos la técnica presenta variaciones de iluminación, expresión, orientación y presencia o falta de componentes faciales como barba o gafas. La imagen también se somete a un pre-procesamiento lo cual implica una normalización del gradiente de iluminación y una ecualización del histograma. Esto reduce el efecto de las sombras y suaviza el contraste de la imagen.

Un clasificador agregado sirve como núcleo de la decisión del sistema. Está compuesto por tres clasificadores de redes Bayesianas. La selección se basa en las soluciones observadas y los compromisos que interese que se tomen. Por ejemplo, cuando el número de clasificadores aumenta la tasa de detección disminuye así como también lo hace la tasa de verdaderos positivos. Después de que la imagen fuese examinada y sus regiones etiquetadas como candidatas a ser caras o no, hay una etapa de post-procesamiento en la cual se seleccionan las regiones con los mayores valores de probabilidad y se marcan como caras.

H. Schneiderman and T. Kanade formulan una técnica diferente en la que se utiliza una vista en 2-dimensiones para representar la geometría de un objeto (cara, coche) en 3-dimensiones [36]. A la vista en 2D se le aplican varias reglas que definirán si un objeto está presente en la vista con respecto a su orientación y pose frente a la cámara. Mediante la aplicación de una transformada de wavelets (ondas ondulatorias), los datos se evalúan estadísticamente y se producen histogramas para representar modelos de variaciones.

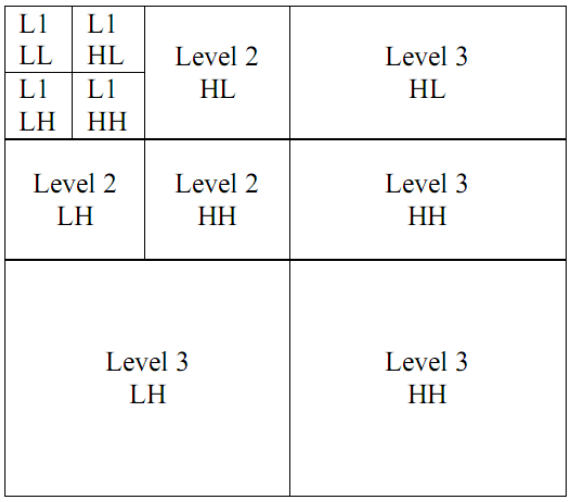


Figura. 2.23. Representación de una imagen mediante ondas ondulatorias

Para poder hacer frente a las variaciones en las imágenes se sigue una estrategia que consta de dos partes. Para las variaciones en pose, se utilizan múltiples detectores estando cada uno de ellos concentrado en una orientación específica (ver Figura 2.24). Para el resto de variaciones, se utiliza un modelo estadístico en cada uno de los detectores. La dificultad de configurar estos modelos estadísticos reside en que desconocemos sus características, y para representarlos han optado por utilizar un producto de histogramas que describa la variación en apariencia de los modelos faciales del resto del mundo físico. Cada histograma se refiere a un atributo visual diferente.

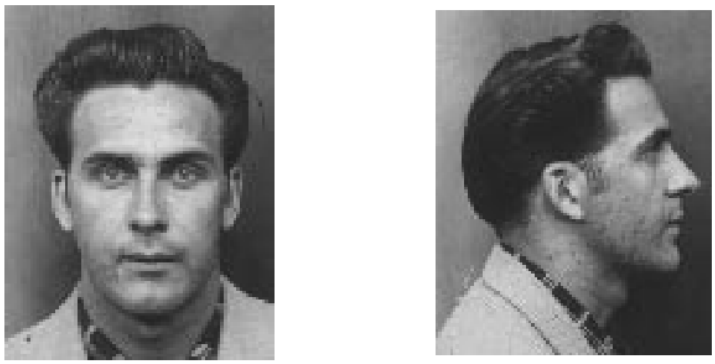


Figura. 2.24. Ejemplos de imágenes de entrenamiento para cada orientación facial.

En el proceso de decisión, un gran conjunto de atributos se utilizan para minimizar el error y para utilizar la mayor cantidad de información posible. Esta técnica trata de modelar conjuntamente a información visual localizada en el espacio, la frecuencia y la orientación. Para ello descompone la apariencia visual a lo largo de estas tres dimensiones. Con el fin de crear los atributos visuales mencionados anteriormente, hace falta seleccionar la información que está localizada a lo largo de estas dimensiones, aplicando una transformada de ondas ondulatorias a la imagen.

La decisión se basa en un escaneado de la imagen de entrada que lleva a la evaluación de atributos de bajo nivel, concentrándose en regiones donde los atributos parezcan prometedores a la hora de encontrar una cara. La clasificación final es un producto de los histogramas que se producen en la etapa de entrenamiento. En este método la principal meta es tratar de utilizar la mayor cantidad de información posible para la decisión.

3 DETECCIÓN ROBUSTA DE CARAS EN TIEMPO REAL

3.1 INTRODUCCIÓN

A lo largo de este capítulo se describe la estructura y el funcionamiento del detector facial de Viola y Jones [1]. Este detector facial servirá de base al trabajo realizado en este proyecto. Se ha escogido debido a las contribuciones realizadas, enumeradas a continuación, así como por su velocidad a la hora de detectar caras en imágenes, sin olvidarnos de la popularidad que ha adquirido gracias a la implementación OpenCV [38].

Viola y Jones construyeron un sistema de detección de la cara frontal que consigue la detección y tasas de falsos positivos comparables a los mejores resultados publicados [33], [36] y [34]. Este sistema de detección de rostros se distingue de la mayoría claramente por su capacidad de detectar rostros muy rápidamente. Operando sobre imágenes 384 por 288 píxeles, se detectan caras a una velocidad de 15 fotogramas por segundo en un Pentium III de 700 MHz.

Este método posee tres principales aportaciones a la detección de rostros:

- Imagen integral, nueva imagen de representación que permite realizar una rápida evaluación de las características.
- Clasificador simple y eficiente que utiliza el método de Adaboost para seleccionar grupos de características, que serán las que se utilicen para llevar a cabo la detección.
- Un método para combinar clasificadores sucesivamente más complejos en una estructura en cascada que aumenta dramáticamente la velocidad del detector, centrándose en las regiones prometedoras de la imagen.

3.2 CARACTERÍSTICAS E IMAGEN INTEGRAL

El sistema de detección de Viola-Jones utiliza grupos de características simples para llevar a cabo la detección en lugar de los pixeles directamente. El uso de estas características posibilita una velocidad mucho mayor a la hora de detectar caras frente a un sistema basado en píxeles y beneficia la codificación en un dominio adecuado.

Viola y Jones utilizan 3 tipos de características en su sistema de detección: basadas en dos rectángulos, basada en tres rectángulos y basada en cuatro rectángulos. Los rectángulos tienen el mismo tamaño y son horizontal o verticalmente adyacentes.

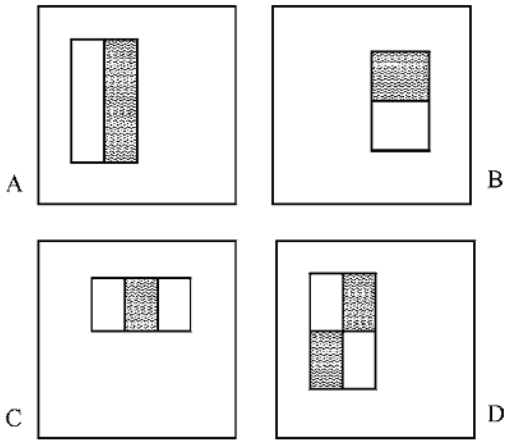


Figura. 3.1. Características utilizadas en el detector Viola-Jones

De manera simplificada, las características pueden ser vistas como evaluaciones de la intensidad de conjuntos de píxeles. La suma de la luminancia de los píxeles en la región blanca se resta de la suma de los píxeles en la región oscura. El valor obtenido mediante la diferencia es el valor de la característica y puede combinarse con otros formando hipótesis en regiones de una imagen.

Un hecho de particular relevancia es la selección del conjunto de características a partir de las imágenes de entrenamiento, en su caso de 24x24 píxeles. El número de características de cada tipo para cada imagen es muy grande, 160.000, y de ellas se eligen las más adecuadas para la detección.

3.2.1 Imagen integral

A la hora de crear un sistema de detección facial resulta crucial encontrar un compromiso entre velocidad y eficiencia. Mediante el uso de una nueva representación de las imágenes, llamada imagen integral, Viola y Jones describen un método de evaluación de características de manera efectiva y a mayor velocidad.

El concepto de la imagen integral es fácilmente comprensible. Esta estructura se construye tomando la suma de los valores de luminancia de los píxeles que se encuentran por encima y a la izquierda de un cierto punto en la imagen. Viola y Jones presentan la imagen integral como la integral doble de una imagen primero a lo largo de las filas y después a lo largo de las columnas. La imagen integral en el punto *(x,y)* viene dada por:

$$ii(x, y) = \sum_{x=x,\ y=y} i(x', y')$$

Donde ii(x,y) es la imagen integral y i(x,y) es el valor de la imagen en unas coordenadas especificas (ver Figura 3.2). Usando el siguiente par de igualdades:

$$s(x, y) = s(x, y-1) + i(x, y)$$
$$ii(x, y) = ii(x-1, y) + s(x, y)$$

(Donde *s(x,y)* es la suma acumulativa, *s(x,-1)*= 0 y $ii(-1, y) = 0$) la imagen integral puede ser calculada en un solo paso sobre la imagen original.

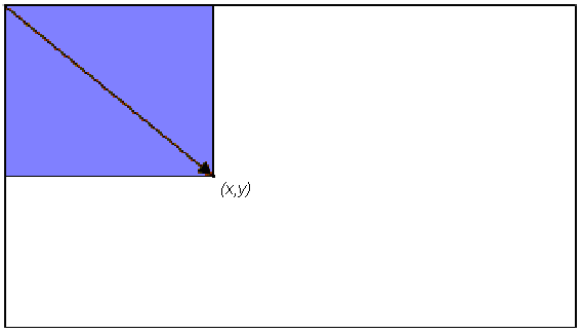


Figura. 3.2. El valor de la imagen integral en un punto (x,y) es la suma de los píxeles de arriba a la izquierda

La importancia de la imagen integral se basa en la capacidad de calcular rápidamente la suma de píxeles dentro de un área determinada de la imagen. Cualquier suma dentro de un área de la imagen puede calcularse utilizando cuatro referencias (ver Figura 3.3). Por lo tanto la diferencia entre dos regiones puede calcularse utilizando 8 referencias dentro de la imagen. Sin embargo, teniendo en cuenta que, por ejemplo los dos primeros tipos de características utilizan dos regiones rectangulares adyacentes la diferencia puede realizarse utilizando 6 referencias, en el caso del tercer tipo se utilizarían 8 referencias y en el cuarto tipo 9 referencias.

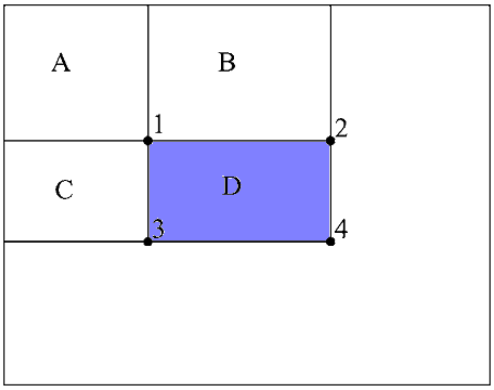


Figura. 3.3. La suma de las píxeles dentro del rectángulo D puede ser calculada con cuatro referencias. El valor de la imagen integral en la ubicación 1 es la suma de los píxeles en el rectángulo A. El valor en la posición 2 es A + B, en el punto 3 es A + C, y en el punto 4 es A + B + C + D. La suma en D se puede calcular como 4 + 1 - (2 + 3).

Conociendo:

$$f * g = \int\int (f' * g')$$

$$(f'') * \left(\int\int g\right) = f * g$$

El cálculo de la suma rectángulo se puede expresar como un producto escalar, $i \cdot r$ donde i es la imagen y la r una imagen con un valor 1 dentro del rectángulo de interés y 0 fuera. Esta operación puede ser reescrita como:

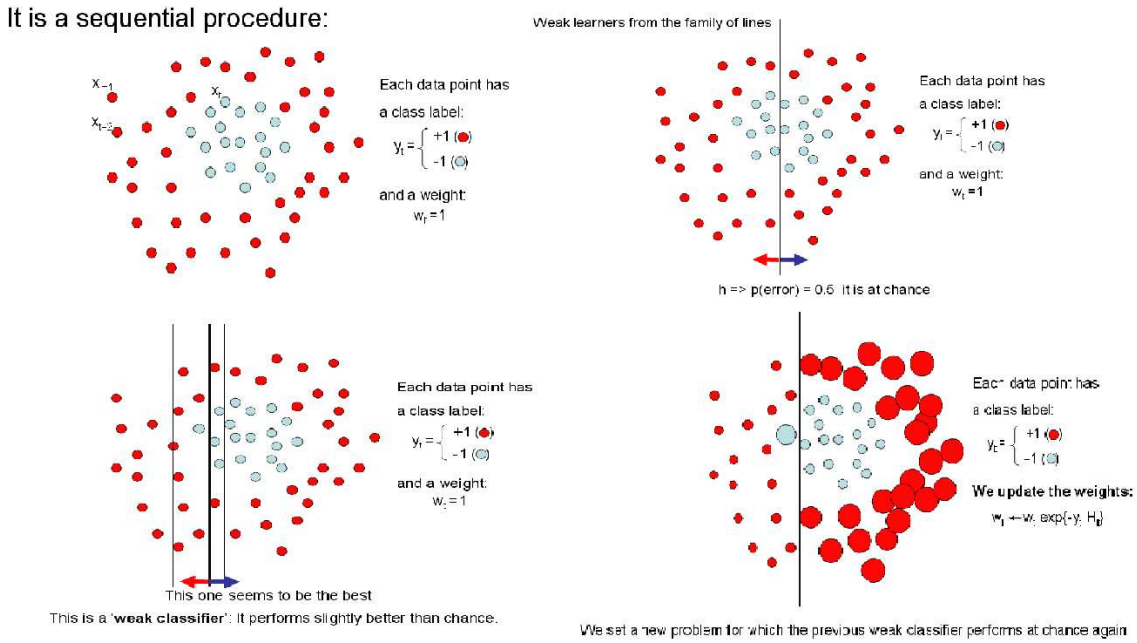
$$i \cdot r = \left(\int \int i\right) \cdot r''$$

La imagen integral es, de hecho, la integral doble de la imagen (primero a lo largo de las filas y luego por columnas).

3.2.2 Clasificador simple

Dado un conjunto de características y un conjunto de entrenamiento de imágenes positivas y negativas, cualquier número de algoritmos de aprendizaje se podrían utilizar para crear una función de clasificación. Recordar que hay 160.000 rectángulos de características asociados con cada subimagen, un número mucho mayor que el número de píxeles. Si bien cada característica se puede calcular de manera muy eficiente, la computación del conjunto completo tiene un costo muy alto. Viola y Jones proponen que un conjunto muy pequeño de estas características se pueden combinar para formar un clasificador eficaz. El reto principal es encontrar estas características. Para ello establecen una variante del algoritmo AdaBoost de Freund y Shaphire [37]. El algoritmo se utiliza para mejorar el rendimiento de un algoritmo de aprendizaje simple. Este algoritmo simple se llama clasificador simple.

En nuestro sistema, una variante de AdaBoost se utiliza tanto como para seleccionar las características como para entrenar el clasificador. En su forma original, el algoritmo de aprendizaje AdaBoost se utiliza para mejorar la clasificación de un algoritmo de aprendizaje simple. Lo hace mediante la combinación de una colección de clasificadores débiles que forman un clasificador más fuerte.



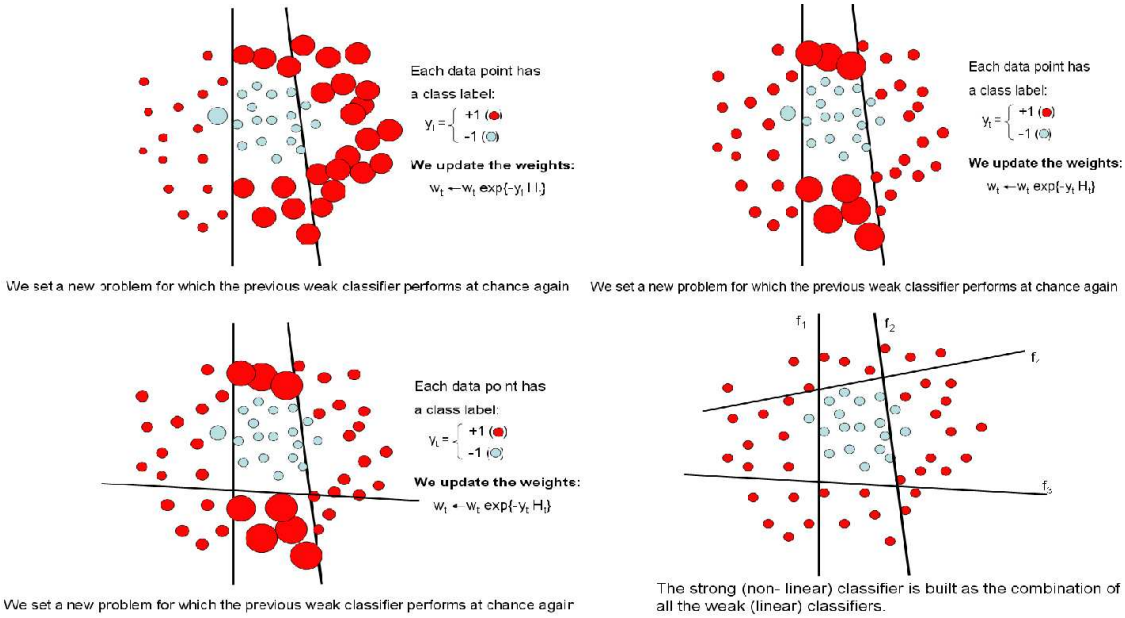


Figura. 3.4. Ejemplo construcción de un clasificador fuerte a partir de clasificadores débiles.

Las garantías proporcionadas por el AdaBoost son bastante fuertes. Freund y Schapire han demostrado que el error de entrenamiento de los clasificadores fuertes se aproxima a cero de manera exponencial con el número de rondas.

El procedimiento convencional AdaBoost puede ser fácilmente interpretado como un codicioso proceso de selección de características. La clave consiste en asociar un peso grande a cada característica de clasificación buena y un peso más pequeño a las pobres.

Un clasificador débil (h(x, f, p, θ)) consiste en una característica (f), un umbral (θ) y una paridad (p). La salida del clasificador es binaria y depende de si el valor de una característica se encuentra por encima o debajo del umbral.

$$h(x, f, p, \theta) = \begin{cases} 1 & si \;\; pf(x) < p\theta \\ 0 & otroscasos \end{cases}$$

Los clasificadores simples que se utilizan pueden verse como nodos de decisión en estructuras en árbol. Teniendo en cuenta la gran cantidad de características en una imagen de 24x24 píxeles, el algoritmo AdaBoost debe seleccionar las características que mejor diferencien entre caras y no caras dentro de un conjunto de imágenes, y en él reside la mayor parte del trabajo del entrenador.

La clasificación es un componente muy importante en los sistemas inteligentes. El problema que se plantea a la hora de realizar una clasificación consiste en decidir a qué “clase” pertenece un objeto. Sobre dicho objeto se realizan varias mediciones, que son las llamadas “características”, en las que se basamos a lo largo del proyecto. Por ello, lo que interesa es aprender y encontrar una relación entre dichas características y las diversas clases a las que se enfrentan. La selección de características se implementa del siguiente modo para cada ronda de boosting:

i. Evaluar cada característica, aplicando cada filtro rectangular, sobre cada imagen de ejemplo.
ii. Ordenar las imágenes según los valores obtenidos en el paso anterior.
iii. Seleccionar el mejor umbral para cada característica.

iv. Seleccionar el mejor filtro/umbral, es decir, la mejor característica.
v. Actualizar los pesos (importancias).

3.2.3 Algoritmo AdaBoost

- Dadas imágenes de ejemplo $(x_1, y_1),\ldots,(x_n, y_n)$ donde $y_i = 0,1$ para ejemplos negativos o positivos de caras respectivamente, y T el número de hipótesis a construir.
- Inicializa los pesos $\omega_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0,1$ respectivamente, donde m y l son el número de ejemplos negativos y positivos respectivamente.
- Para t = 1,…,T :

1. Normalizar los pesos, $\omega_{t,i} \leftarrow \frac{\omega_{t,i}}{\sum_{j=1}^{n} \omega_{t,j}}$
2. Seleccionar el mejor clasificador simple respecto al error ponderado

$$\varepsilon_t = \min_{f,p,\theta} \sum_i \omega_i \left| h(x_i, f, p, \theta) - y_i \right|$$

3. Definir $h_t(x) = h(x, f_t, p_t, \theta_t)$ donde f_t, p_t, θ_t son los valores que minimizan ε_t.
4. Actualizar los pesos:

$$\omega_{t+1,i} = \omega_{t,i} \beta_t^{1-e_i}$$

donde $e_i = 0$, si la imagen x_i está correctamente clasificada, o $e_i = 1$ en caso contrario. $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$.

5. El clasificador final sería:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otroscasos \end{cases}$$

donde $\alpha_t = \log \frac{1}{\beta_t}$

Figura. 3.5. Algoritmo AdaBoost

El algoritmo Adaboost, se utiliza para seleccionar los mejores clasificadores simples. Teniendo en cuenta que existe un clasificador simple por cada característica hay que evaluar un total de K·N clasificadores siendo K el número de características por imagen y N el número de imágenes que contenga el conjunto de entrenamiento.

La ventaja principal de AdaBoost es su velocidad de aprendizaje. En este algoritmo los pesos se asignan de manera que favorezcan la clasificación de las caras, consiguiendo que estos ejemplos tengan mayores pesos o importancia. Las mejores características se eligen basándose en el error ponderado que se produce. Este error ponderado es una función que utiliza los errores pertenecientes a los ejemplos de entrenamiento. El peso de un ejemplo clasificado correctamente se modifica, aumenta, mientras que el peso de un ejemplo mal clasificado se mantiene constante. Con esto se consigue que sea más difícil que la segunda característica clasifique erróneamente un ejemplo que haya sido clasificado erróneamente por la primera característica frente a un ejemplo clasificado correctamente por esa primera característica. Otra manera de ver esto sería que la segunda característica, y las sucesivas, se ven forzadas a tener más en cuenta los ejemplos clasificados erróneamente por características anteriores.

Después de sucesivas iteraciones del algoritmo el resultado es un conjunto de hipótesis ponderado que conforma un clasificador fuerte. La hipótesis final encontrada después de T iteraciones se obtiene al concluir el algoritmo, donde la clasificación binaria (cara Vs. no-cara) se realiza en función de los pesos individuales α (representando los umbrales) y la suma del producto de cada peso y la clasificación de cada ejemplo en particular $\alpha_t h_t(x)$.

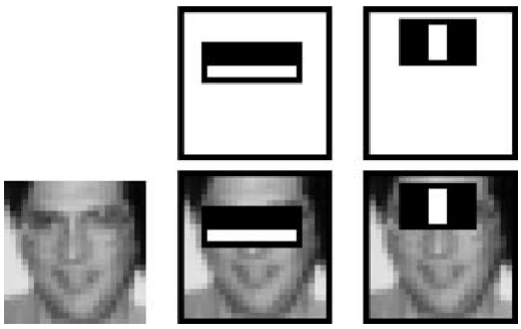


Figura. 3.6. Las dos características se muestran en la primera fila y luego superpuestas en un rostro en la fila inferior. La primera de las medidas diferencia de intensidad entre la región de los ojos y la región de las mejillas. La segunda característica compara la intensidad en las regiones de los ojos con la del puente de la nariz.

3.3 CLASIFICADOR EN CASCADA

El principio básico del algoritmo de detección facial de Viola y Jones consiste en escanear el detector muchas veces a través de una misma imagen, en diferentes posiciones y a distintas escalas. Incluso si una imagen contiene muchas caras está claro que la mayoría de las sub-ventanas que se escaneen no contendrán ninguna cara.

Esto lleva a una nueva manera de ver el problema: En vez de encontrar caras, el algoritmo debería descartar no-caras. La idea subyacente se basa en que es más fácil descartar una imagen que no contenga una cara que encontrar una cara en una imagen. Con esto en mente, parece ineficiente construir un detector que contenga un solo clasificador fuerte ya que el tiempo de clasificación será constante sin importarnos la entrada del clasificador, ya que el clasificador tiene que evaluar todas las características que lo forman. Aumentar la velocidad de clasificación generalmente implica que el error de clasificación aumentará inevitablemente, ya que para disminuir el tiempo de clasificación se debería disminuir el número de clasificadores simples que se utilizan. Para evitar esto Viola y Jones proponen un método para reducir el tiempo de clasificación manteniendo los requerimientos de rendimiento del clasificador.

Este método consiste en el uso de una cascada de clasificadores fuertes. El trabajo en cada etapa del clasificador consiste en determinar si la sub-ventana que se analiza es definitivamente una no-cara o podría ser una cara. Cuando una sub-ventana es clasificada como no cara en alguna de las etapas del detector, se descarta inmediatamente. En caso contrario, si se clasifica como una posible cara, pasa a la siguiente etapa del clasificador. Se identificará una sub-ventana como contenedora de una cara si y sólo si pasa a través de todas las etapas del detector de forma positiva.

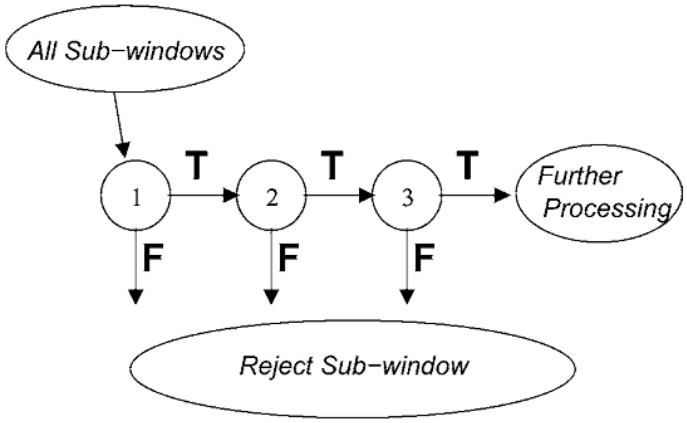


Figura. 3.7. Estructura de clasificadores en cascada.

Si se utilizase un clasificador que tuviese un único estado, normalmente habría que aceptar los falsos negativos para reducir la tasa de falsos positivos. Sin embargo, para las primeras etapas del clasificador en cascada se acepta una alta tasa de falsos positivos esperando que las etapas posteriores puedan encargarse de reducir esta tasa mediante clasificadores más especializados. Con esto se pretende también reducir la tasa de falsos negativos en el clasificador final, ya que una sub-ventana será clasificada como cara sólo en el caso de que haya pasado por todas las etapas del clasificador.

3.3.1 Entrenamiento de los clasificadores en cascada

El proceso de diseño de la cascada empleado por Viola y Jones se basa en objetivos de detección y rendimiento similares a lo que se hizo hasta el momento. Los sistemas anteriores obtenían tasas de detección de entre el 85% y el 95% y tasas de falsos positivos extremadamente bajas del orden de 10^{-5}. Es por ello que el sistema debería tener un número de etapas suficientes para obtener resultados similares.

Dada una cascada de clasificadores, la tasa de falsos positivos de la cascada es:

$$F = \prod_{i=1}^{K} f_i$$

Donde F es la tasa de falsos positivos del detector, K es el número de clasificadores, y f_i es la tasa de falsos positivos calculada para el clasificador i. Por otro lado la tasa de detección de la cascada se calcularía según:

$$D = \prod_{i=1}^{K} d_i$$

Donde D es la tasa de detección del detector, K es el número de clasificadores, y d_i es la tasa de detección calculada para el clasificador i. Una vez establecido esto, vemos como por ejemplo un detector consistente en 10 etapas con una tasa de detección del 99% por etapa y una tasa de falsos positivos del 30% por clasificador nos llevaría a una tasa de detección global del $0.99^{10} \approx 0.9$ y una tasa de falsos positivos global del $0.3^{10} \approx 6x10^{-6}$.

El número de características evaluadas cuando se escanean imágenes reales es un proceso probabilístico. Cualquier sub-ventana pasará a lo largo de la cascada, un clasificador cada vez, hasta que se decida si la ventana es negativa o, en raras circunstancias, positiva. La medida clave de cada clasificador es la tasa de positivos, la proporción de ventanas que hayan sido etiquetadas como positivas (pueden contener una cara). El número esperado de características que se evalúan son:

$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_j \right)$$

Donde N es el número esperado de características evaluadas, K el número de clasificadores, p_i la tasa de positivos del i-ésimo clasificador y n_i el número de características en el i-ésimo clasificador.

El proceso de entrenamiento debe considerar las limitaciones a las que se ve sometido. En la mayoría de los casos, los clasificadores construidos utilizando más características tendrán una mayor tasa de detección y una menor tasa de falsos positivos. Al mismo tiempo, los clasificadores con más características necesitarán más tiempo para determinar si una sub-ventana contiene o no una cara. A la hora de entrenar el clasificador uno debe optimizar el número de etapas del clasificador, el número de características, n_i, de cada etapa y el umbral de cada etapa. Sin embargo, realizar esta optimización es un trabajo tremendamente costoso.

Para simplificar este problema los autores han realizado un algoritmo para poder entrenar de manera efectiva la cascada de clasificadores (Figura 3.8). En este caso el usuario elige las tasas de falsos positivos y de detección de cada etapa así como la tasa de falsos positivos referente a todo el detector. Cada etapa del detector se entrena utilizando AdaBoost del modo descrito en la Figura 3.5, incrementando el número de características de la etapa hasta que se obtengan las tasas de falsos positivos y de detección deseadas. Dichas tasas se determinan confrontando el detector con un set de validación. Si la tasa de falsos positivos global no es la que interesa se añade otra etapa al clasificador.

- El usuario selecciona el valor de f, la tasa de falsos positivos máxima aceptable por etapa, y de d, tasa de detección mínima aceptable por etapa.
- El usuario selecciona la tasa de falsos positivos global para el detector, F_{target}.
- P = conjunto de ejemplos positivos.
- N = Conjunto de ejemplos negativos.
- $F_0 = 1.0$; $D_0 = 1.0$
- i = 0
- mientras $F_i > F_{target}$
  - $i \leftarrow i + 1$
  - $n_i = 0$; $F_i = F_{i-1}$
  - mientras $F_i > f \times F_{i-1}$
    * $n_i \leftarrow n_{i+1}$
    * Utilizar P y N para entrenar un clasificador con n_i características utilizando AdaBoost.
    * Evaluar el clasificador en cascada actual sobre el conjunto de validación para determinar F_i y D_i.
    * Disminuir el umbral para el clasificador i-ésimo hasta que el actual clasificador en cascada tenga una tasa de detección de al menos dx D_{i-1} (esto también afecta a F_i).
  - $N \leftarrow \varnothing$
  - Si $F_i > F_{target}$ evaluar el detector en cascada actual utilizando el conjunto de no-caras y poner todos los falsos positivos en el set N.

Figura. 3.8. Algoritmo de entrenamiento para construir un detector en cascada.

Por otro lado, con el objetivo de demostrar que el clasificador en cascada es más veloz frente a un solo clasificador fuerte y, además, no pierde calidad en la detección, se entrenó un clasificador con una etapa de 200 características por un lado, y otro con 10 etapas de 20 características cada una. La primera etapa del clasificador en cascada fue entrenada usando 5000 caras y 10000 no-caras, escogidas de forma aleatoria a partir de imágenes que no contengan caras. La segunda etapa del clasificador fue entrenada en las mismas 5000 caras, más 5000 falsos positivos de la primera etapa del clasificador. Este proceso continuó de manera que las posteriores etapas fueron entrenadas con los falsos positivos de las etapas anteriores. El clasificador fuerte de 200 características se entrenó con todos los ejemplos utilizados para entrenar todas las etapas del clasificador en cascada.

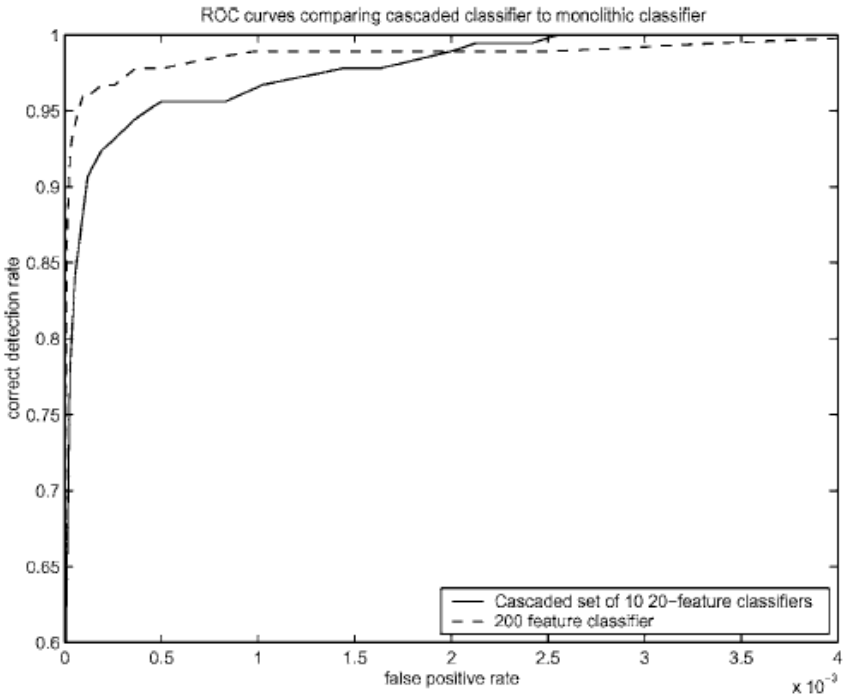


Figura. 3.9. Curvas ROC comparando un clasificador con una etapa de 200 características y otro con 10 etapas de 20 características cada una. La precisión no es significativamente diferente, pero la velocidad del clasificador en cascada es casi 10 veces mayor.

En la Figura 3.9 se presentan las curvas ROC comparando la eficiencia de los dos clasificadores. Esto demuestra que existe poca diferencia entre los dos en términos de precisión. Sin embargo, existe una gran diferencia en términos de velocidad. El clasificador en cascada es casi 10 veces más rápido, desde su primera desecha la mayoría de no-caras de manera que en las siguientes etapas nunca serán evaluadas.

3.4 RESULTADOS

En esta sección se dará una breve visión de los resultados obtenidos por Viola y Jones según su publicación [1]. A lo largo de este apartado se comentará el conjunto de entrenamiento utilizado, la estructura y la formación del detector en cascada y los resultados del procesado de imágenes de diferentes tamaños, donde haya que integrar múltiples detecciones en diferentes localizaciones y a diferentes escalas.

3.4.1 Datos de entrenamiento

El conjunto de entrenamiento consiste en 4916 caras con una resolución de 24x24 píxeles y etiquetadas a mano. Algunas de esas imágenes se muestran en la Figura 3.10 Viola y Jones remarcan la diferencia entre su conjunto de entrenamiento y el utilizado por Rowley [33] el cual usaba imágenes de 16x16 píxeles. Presumiblemente con los ejemplos de Viola y Jones se obtienen mejores resultados y la información contenida en estas imágenes puede ser utilizada para descartar no-caras en etapas previas de la cascada.

Figura. 3.10. Ejemplo de imágenes frontales utilizadas por Viola y Jones

3.4.2 Estructura del detector en cascada

El detector final obtenido por Viola y Jones contiene una cascada de 38 etapas con un total de 6060 características. El primer clasificador en cascada utiliza dos características y rechaza aproximadamente el 50% de las no-caras mientras que detecta correctamente casi el 100% de las caras. El siguiente clasificador tiene 10 características y rechaza el 80% de las no-caras detectando casi el 100% de las caras. Las siguientes dos etapas tienen clasificadores de 25 características y las siguientes 3 etapas tienen clasificadores de 50 características y así sucesivamente.

Es interesante remarcar que Viola y Jones eligieron de forma manual el número de características de los 7 primeros estados para reducir el tiempo de entrenamiento del detector, lo que nos hace ver que a pesar de la velocidad que pueda tener el detector a la hora de trabajar, su entrenamiento es tremendamente costoso, con lo que los mismos autores reconocen haber modificado ligeramente el algoritmo presentado en la Figura 3.9 para facilitar la tarea computacional [1]. Las primeras imágenes de no-caras utilizadas para el entrenamiento del primer nivel de la cascada se obtuvieron mediante la selección aleatoria de un conjunto de 9500 imágenes que no contienen caras. Los ejemplos de no-cara utilizados para las etapas posteriores se obtuvieron mediante el análisis parcial de la cascada a través de imágenes no-cara y recogida de falsos positivos. Un máximo de 6000 no-caras fueron recolectadas para cada capa. Hay aproximadamente 350 millones de imágenes no-cara de las 9500 imágenes iniciales. El tiempo de entrenamiento utilizado para el detector con sus 38 etapas en una máquina 466MHz AlphaStation XP900 fue del orden de semanas.

3.4.3 Procesado de imágenes

Todos los ejemplos utilizados para el entrenamiento eran de imágenes normalizadas con el objetivo de minimizar el efecto de las diferentes condiciones de iluminación. Por lo tanto, parece lógico entender que la normalización es también necesaria para la detección.

Por otro lado el detector se aplica sobre imágenes de diversos tamaños y las caras que aparecen en las mismas contienen también distintos tamaños. Por ello el sistema debe ser capaz de extraer sub-ventanas de una imagen y analizarlas. Según los autores se obtienen buenos resultados escalando el tamaño de estas sub-ventanas con un factor de 1.25. El detector también escanea una imagen en diversas localizaciones, con lo que las sub-ventanas se mueven a lo largo de la imagen un número determinado de píxeles, Δ. La elección de este número afectará tanto a la velocidad del detector como a su rendimiento. Dado que las imágenes de entrenamiento tienen una cierta variabilidad de traslación el detector logra un rendimiento de detección bueno a pesar de pequeños cambios en la imagen. Como resultado, el detector de sub-ventanas puede desplazarse más de un píxel cada vez. Sin embargo, desplazamiento de más de un píxel tiende a disminuir la tasa de detección y el número de falsos positivos ligeramente. En la publicación de Viola y Jones se muestran resultados utilizando tanto Δ = 1, como Δ = 1.5, utilizando escalas de 1 y 1.25 respectivamente (Figura 3.11).

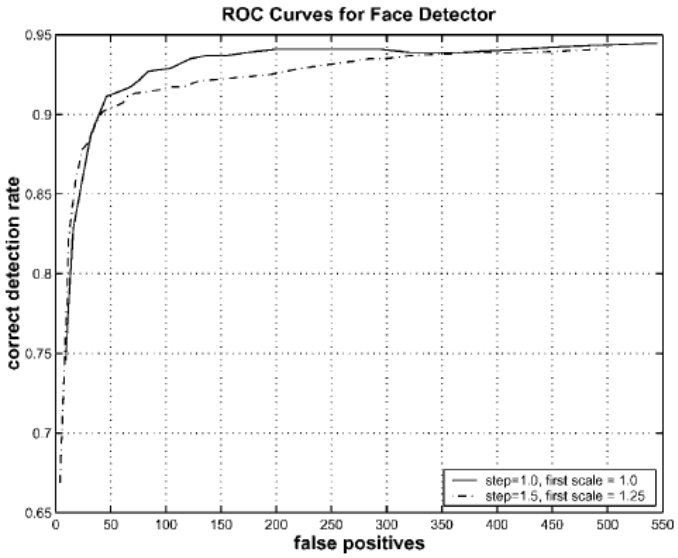


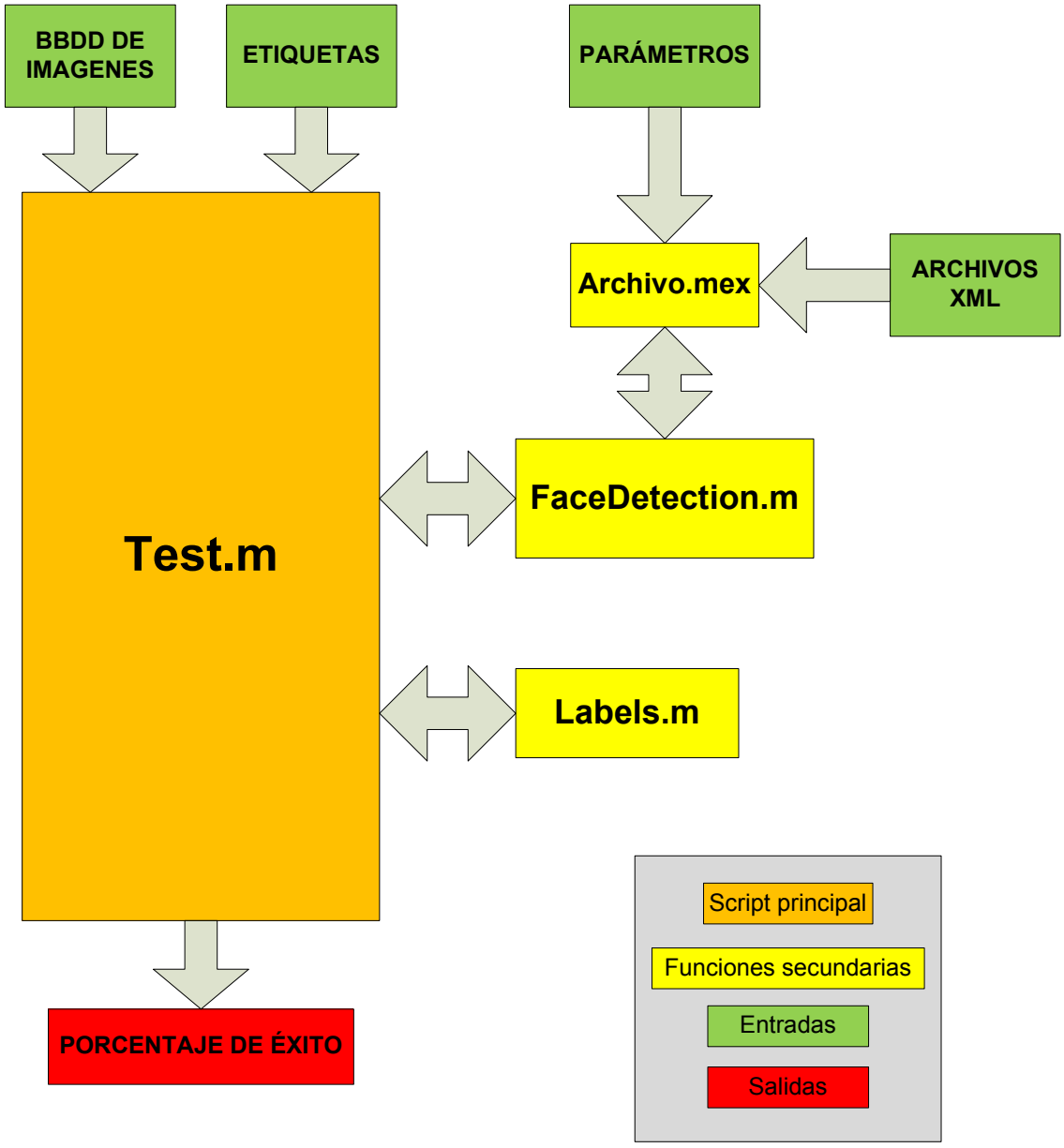
Figura. 3.11. Curvas ROC para detector de caras de Viola y Jones sobre la base de datos MIT + CMU.

Dado que el detector final no es sensible a pequeños cambios en la traslación y en la escala, por lo general se producirán varias detecciones alrededor de cada cara en una imagen. En la práctica, a menudo tiene sentido hacer una detección final por cara. Con este fin, es útil un post-proceso a las sub-ventanas detectadas con el fin de combinar las detecciones superpuestas en una única detección. En estos experimentos las detecciones se combinan de una manera muy simple. Primero se divide el conjunto de las detecciones en subconjuntos disjuntos. Dos detecciones están en el mismo subconjunto si sus regiones de delimitación se superponen. Cada partición produce una única detección final. Las esquinas de la región de delimitación definitiva son el promedio de las esquinas de los recuadros de todas las detecciones sobre una misma cara.

4 IMPLEMENTACIÓN DEL ALGORITMO

4.1 DISEÑO FUNCIONAL

4.1.1 Esquema funcional

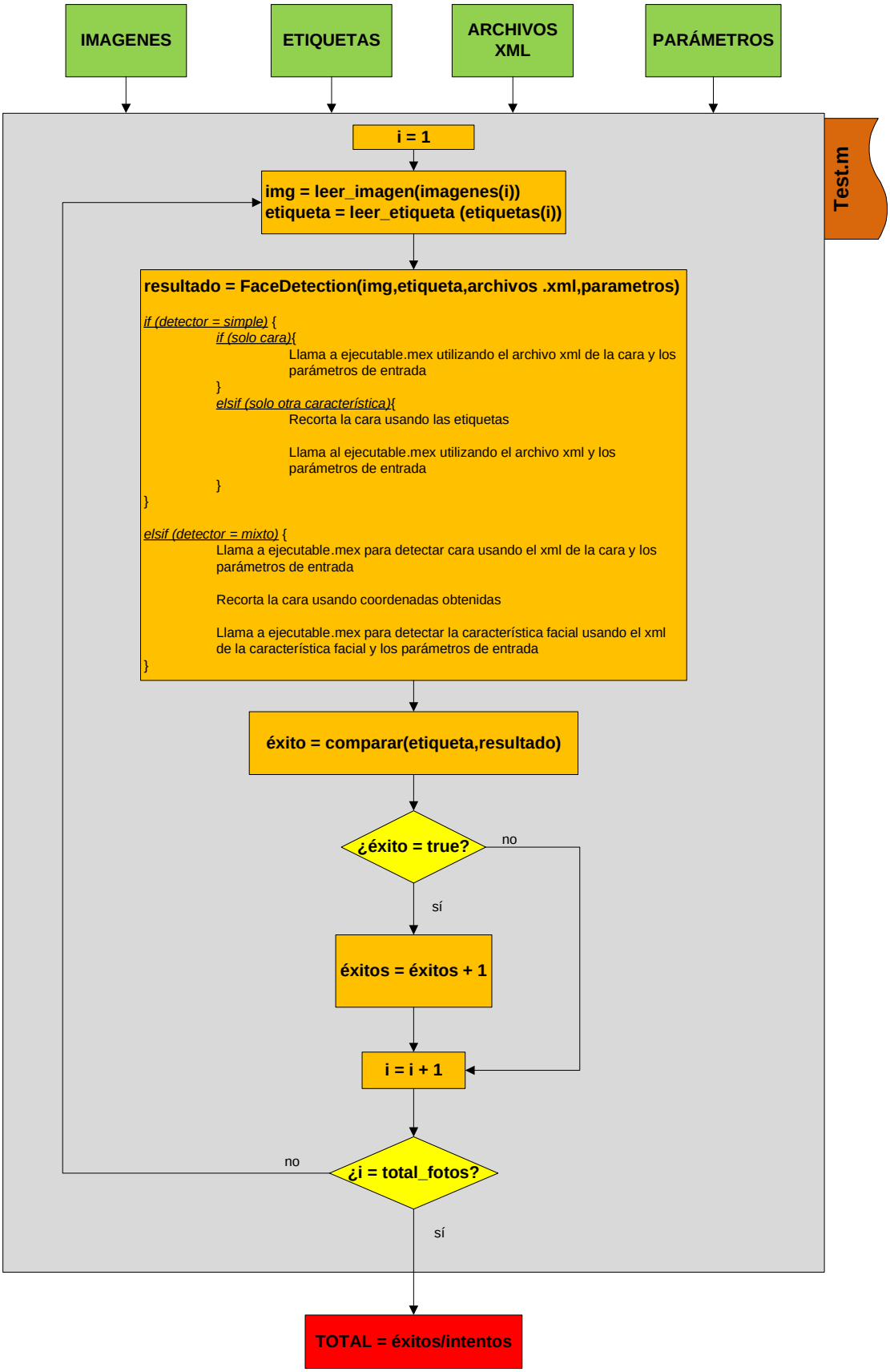


4.1.2 Descripción a alto nivel

El script principal Test.m está diseñado para probar la efectividad del detector de caras y características faciales incluido en el programa FaceDetection.m. Este script se encargará de probar el algoritmo de detección con cada una de las imágenes incluidas en la base de datos de prueba utilizando diferentes combinaciones de parámetros y archivos xml. Posteriormente comparará los resultados obtenidos con los datos reales incluidos en las etiquetas y calculará el porcentaje de éxito para cada combinación de parámetros.

4.2 DISEÑO TÉCNICO

4.2.1 Diagrama de flujos



Test tiene como entradas las imágenes de prueba, las etiquetas de éstas, los archivos xml y los parámetros de cambio. En primer lugar, Test lee una imagen y sus etiquetas. A continuación llama a la función FaceDetection donde se trata la imagen de manera diferente dependiendo de si el detector es simple (detección de cara o característica facial) o mixto (detección de cara y característica /s facial).

Si el detector es simple y lo que se detecta es una cara, FaceDetection llama al ejecutable mex utilizando el archivo xml correspondiente a la cara y los parámetros de entrada. Si el detector es simple y se detecta una característica, se recorta la imagen original utilizando las etiquetas y se llama al ejecutable mex que, en este caso, utiliza el archivo xml y los parámetros de entrada sobre la imagen recortada. Finalmente, si el detector es mixto, FaceDetection llama al ejecutable mex usando el archivo xml y los parámetros para detectar la cara, recorta la cara de la imagen utilizando las coordenadas obtenidas y vuelve a llamar al ejecutable mex para detectar la o las caracteristicas faciales.

Una vez realizada la detección se compara con las etiquetas reales, si el resultado es correcto se almacena como tal y se lee la siguiente imagen. Si el resultado no es correcto se lee directamente la siguiente imagen. Esta operación se realiza hasta que se hayan leído todas las imágenes de la base de datos de prueba. Al final de toda la lectura se hace un recuento del número de éxitos y fracasos.

4.2.2 Funciones en MATLAB

4.2.2.1 FaceDetection.m

FaceDetection.m es una función para la detección de caras y características faciales que devuelve las posiciones de los rectángulos que contienen los objetos detectados. La detección puede ser simple (solo la cara o solo una característica facial) o mixta (la cara junto con una o varias características faciales). Las características faciales detectables por la función son el ojo derecho, el ojo izquierdo, la nariz, la boca y el área ocular.

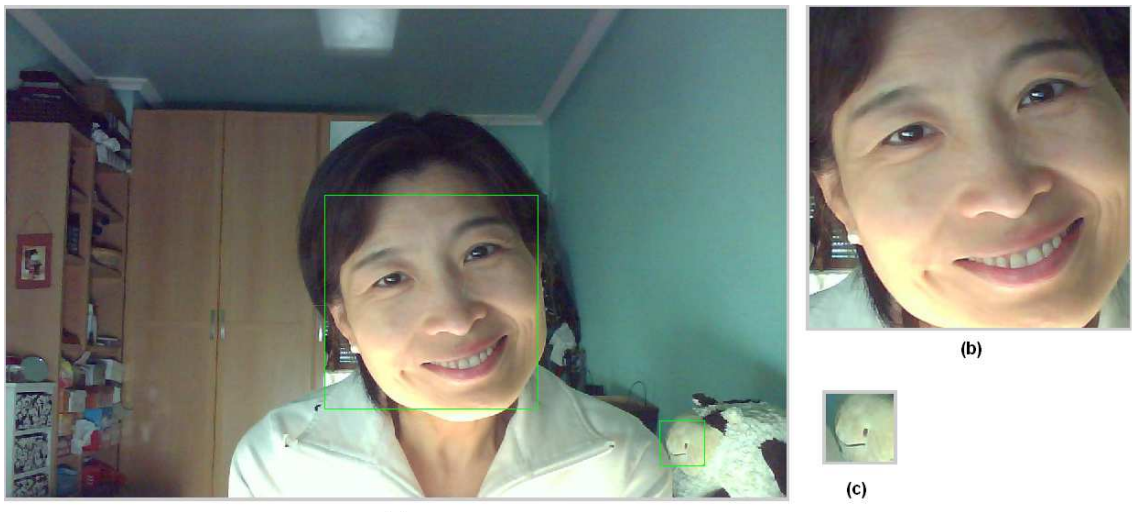
La detección de la cara se realiza sobre la imagen original mientras que la detección de las características faciales se realiza sobre otra imagen recortada. A esa imagen recortada se le va a dar el nombre de Croppedimg.

(a) (b)

Figura. 4.1. (a) Imagen original (b) Imagen recortada, croppedimg.

En el caso de que la detección de características faciales sea simple, Croppedimg se obtiene utilizando las posiciones definidas en FacePositions. De esa manera queda asegurado que el testeo del algoritmo de detección de características faciales se realiza sobre una imagen recortada que se corresponde con una cara.

Si en cambio el detector es mixto, la detección de la cara (y por tanto el recorte de Croppedimg) se realiza utilizando el algoritmo de detección de caras. El detector puede devolver varios candidatos a cara por lo que la detección de las características faciales se realiza para cada candidato. Habrá tantas Croppedimg distintas como candidatos a cara. La imagen recortada debe estar en escala de grises por lo que se realizará la conversión siempre que sea necesaria.



(b)

(c)

(a)

Figura. 4.2. (a) Imagen original con dos candidatos a cara. (b) y (c) Imágenes recortadas de los candidatos a cara, croppedimg.

La llamada desde matlab al detector es realizada desde esta función con la siguiente instrucción:

```
Face = cvod200uint_v02(Array{1}.xml,Img,1,0,60);
```

En Face se devuelven las coordenadas de los rectángulos candidatos a cara o característica facial. Array{i}.xml es el archivo xml correspondiente para detectar la cara o característica facial determinada, siendo distinto en cada caso. Img es la imagen donde vamos a detectar ,1 es la escala, 0 el nivel de depuración y 60 el tamaño mínimo de la ventana donde se realizan las búsquedas. Estos últimos parámetros son los parámetros de entrada del detector y son los que permiten variar algunas de sus características.

4.2.2.2 Labels.m

Labels.m es la función encargada de leer los archivos txt de las posiciones contenidos en *LabelsFace* (descrito en el apartado 4.3.2.1), y coloca cada archivo en una matriz. La función es llamada desde MATLAB de la siguiente manera:

```
MatrizPositions=labels(i,NumImages)
```

La función *Labels.m* abre la carpeta contenedora de los *LabelsFace.txt* y lee el archivo de texto correspondiente al sujeto actual y lo guarda como una matriz (*MatrizPositions*). Por ejemplo si estamos estudiando al sujeto 72 *Labels.m* abre *LabelsFace* y lee *LabelsFacef72.txt* y lo guarda como una matriz que devuelve en *MatrizPositions*.

4.2.2.3 Test.m

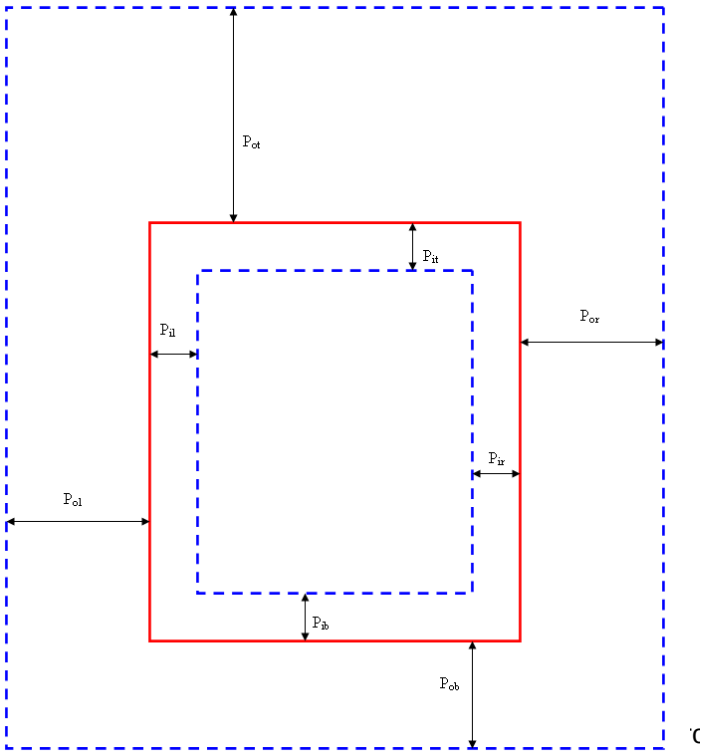
Test.m es el script principal de la ejecución en MATLAB. Se encarga de leer cada una de las imágenes entrantes al programa para posteriormente llamar al detector de caras y características faciales y comprobar su validez, comparando los resultados obtenidos con los valores reales etiquetados anteriormente a mano.

La exactitud se denomina *ACC* y se calcula según el número de *False Positives (FP), False Negatives (FN), True Positives (TP) y True Negatives (TN)* obtenidos al hacer la comparación con las etiquetas. (Estos parámetros quedan definidos en el apartado 5.3)

Test.m va abriendo cada carpeta contenedora de las imágenes (para el sujeto 32 abrirá la carpeta denomina f32 que contendrá todas las imágenes de este sujeto) y las va leyendo y tratando una a una. Finalmente, para cada combinación de parámetros de entrada determinada, calculará el valor de ACC obtenido tras procesar todas las imágenes de la base de datos de prueba.

Como se ha visto en el esquema funcional del apartado 4.1.2, el script principal llama fundamentalmente a dos funciones: *FaceDetection* y *Labels*. La primera se encarga de hacer la detección y devolver los valores de las coordenadas de los lugares donde ha encontrado la cara o la característica facial. Por otra parte, la función *Labels* almacena en la variable *MatrizPositions* las coordenadas asociadas a cada etiqueta existente en la base de datos. De esa manera se puede proceder a la comparación entre las etiquetas reales y los datos obtenidos por la detección.

A partir de los datos de *MatrizPositions,* se toman rectángulos de referencia que servirán para calcular unos márgenes para la cara o características faciales relativos a la dimensión de dicho rectángulo (Figura 4.3.)



P_{it}= porcentaje interior superior
P_{ib}= porcentaje interior inferior
P_{il}= porcentaje interior izquierdo
P_{ir}= porcentaje interior derecho
P_{ot}= porcentaje exterior superior
P_{ob}= porcentaje exterior inferior
P_{ol}= porcentaje exterior izquierdo
P_{or}= porcentaje exterior derecho

Figura. 4.3. Línea azul punteada: márgenes. Línea roja: rectángulo de referencia.

Para un detector no mixto, los rectángulos candidatos que devuelva F*aceDetection* que queden entre los dos márgenes serán considerados como un acierto (TP) y el resto como FP. Si no detecta ningún rectángulo candidato y *MatrizPositions* presenta NaN (no contiene ninguna posición numérica) en las posiciones correspondientes a las coordenadas para el caso actual, se obtendrá un TN. Sin embargo, si no detecta ningún candidato pero sí hay una posición definida tendremos un FN.

En un detector mixto, si no se detectan tantas características como características hubiera en la imagen, se obtiene un FN. Por ejemplo, si estamos detectando la cara, la nariz y la boca y dentro de la *cropedimg* no se detectan al menos una nariz y una boca se obtiene un FN. En el caso de ser detectadas al menos una nariz y una boca, si ninguna nariz o ninguna boca quedan entre los márgenes, se obtendría un FP. Ahora bien, si se detectan al menos una nariz y al menos una boca y por lo menos una nariz y una boca quedan entre los márgenes se logra un TP. La definición del parámetro TN para un detector mixto es la misma que para uno no mixto. (Ver Figuras de la 4.4 a la 4.13)

Cada vez que se obtiene un TP se considera una detección correcta de la cara y se devuelve en pantalla la imagen original con la cara detectada. (Figura 4.13 y 4.14)

Figura. 4.4. TP. Detector simple para nariz. Detección correcta de la nariz. (Línea azul punteada: márgenes. Línea roja: rectángulo de referencia.)

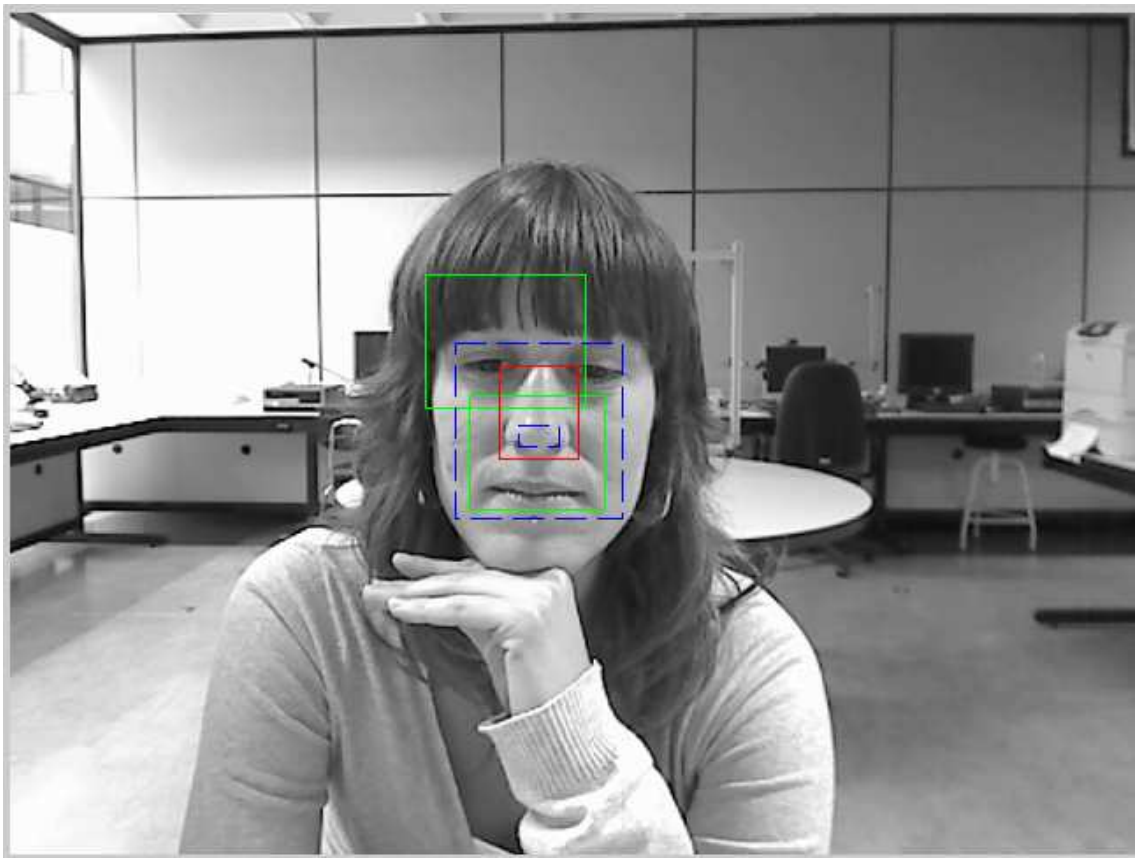


Figura. 4.5. FP y TP. Detector simple de nariz. Detecta 2 candidatos a nariz, uno queda entre los márgenes (TP) y el otro no (FP).

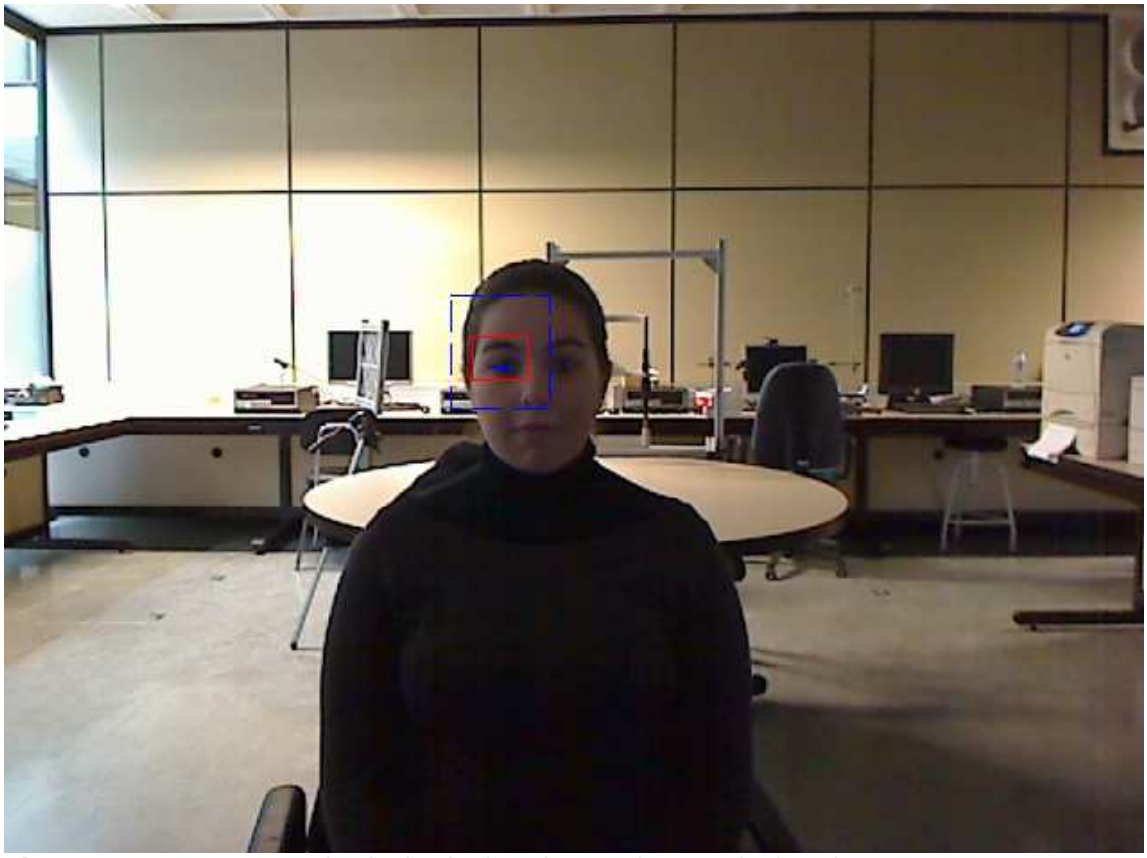


Figura. 4.6. FN. Detector simple de ojo derecho. No detecta ojo derecho.

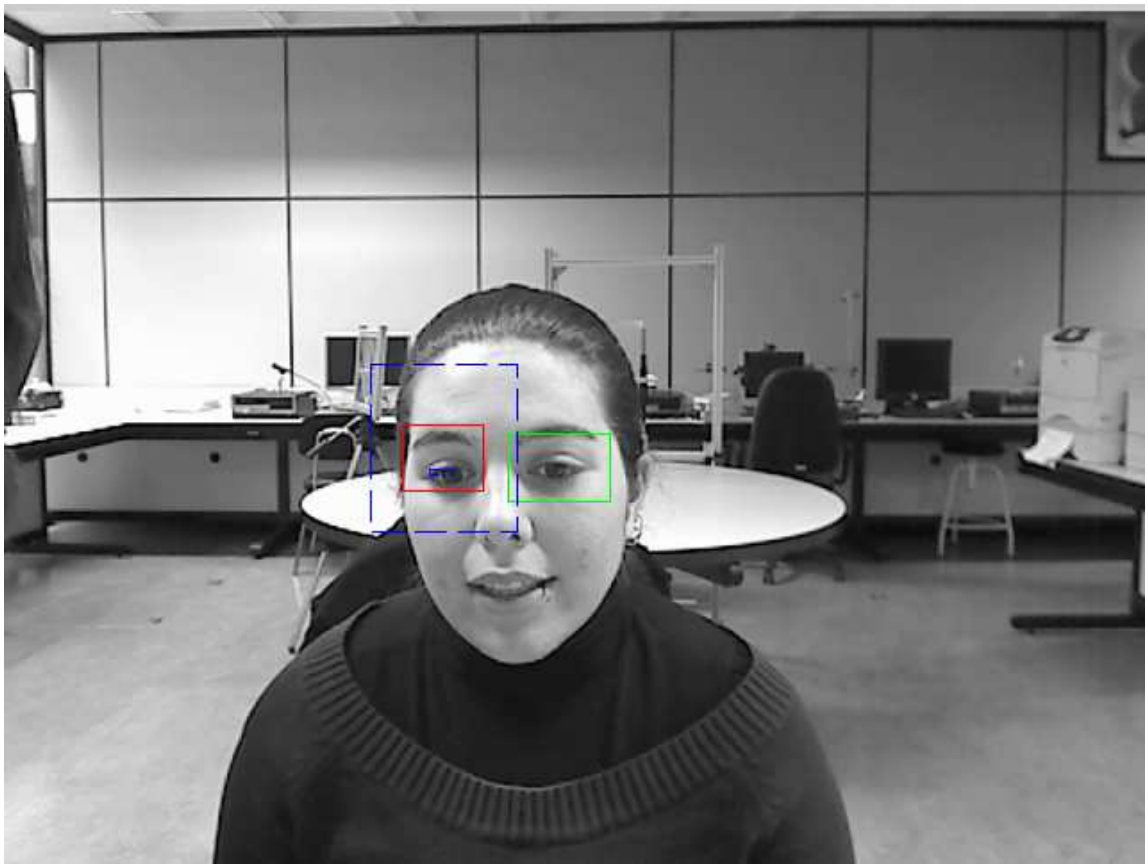
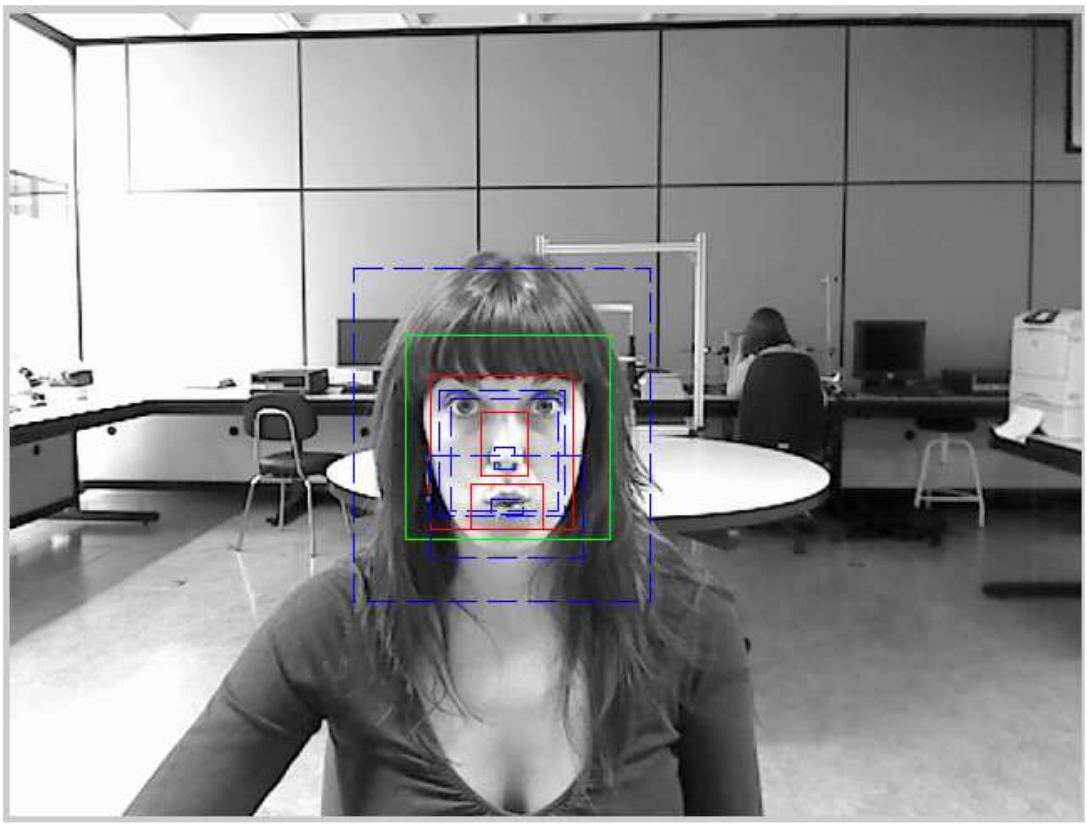


Figura. 4.7. FP. Detector simple de ojo derecho. Detecta un candidato a ojo derecho pero no pertenece a los márgenes.



(a)

(b)

Figura. 4.8. FN. Detector mixto de cara, nariz y boca. (a) No detecta ni nariz ni boca. (b) Detecta nariz pero no boca. (Línea azul punteada: márgenes. Línea roja: rectángulo de referencia.)

Figura. 4.9. FP. Detector mixto de cara, nariz y boca. Se detecta una nariz y una boca pero la boca no pertenece a los márgenes.

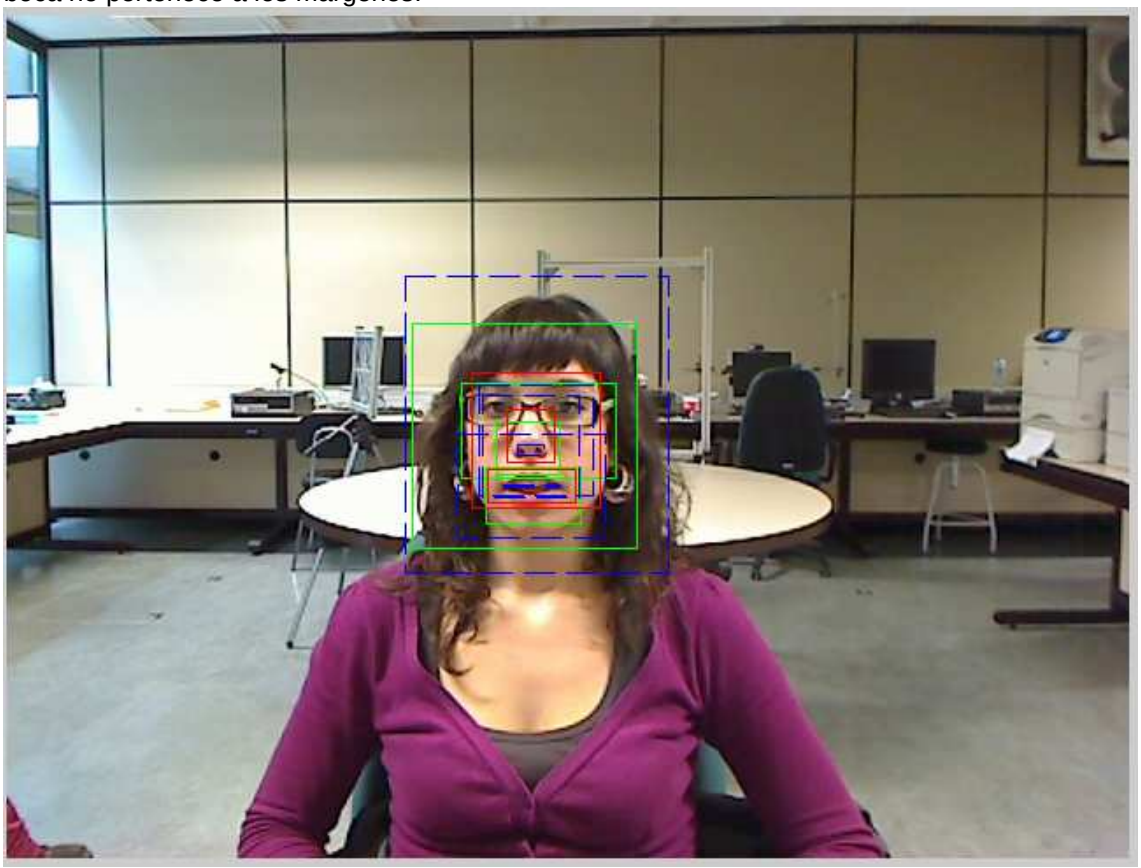
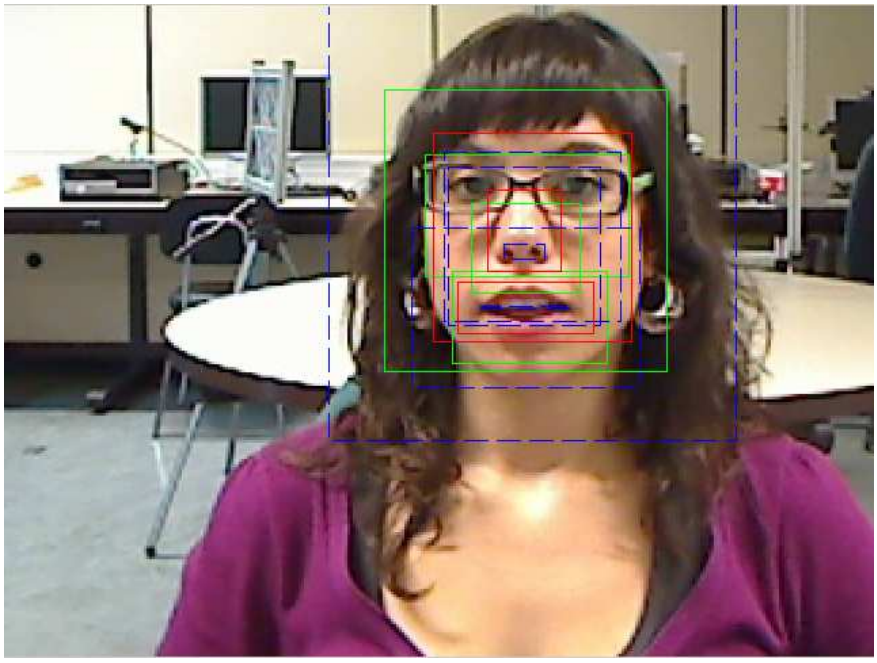
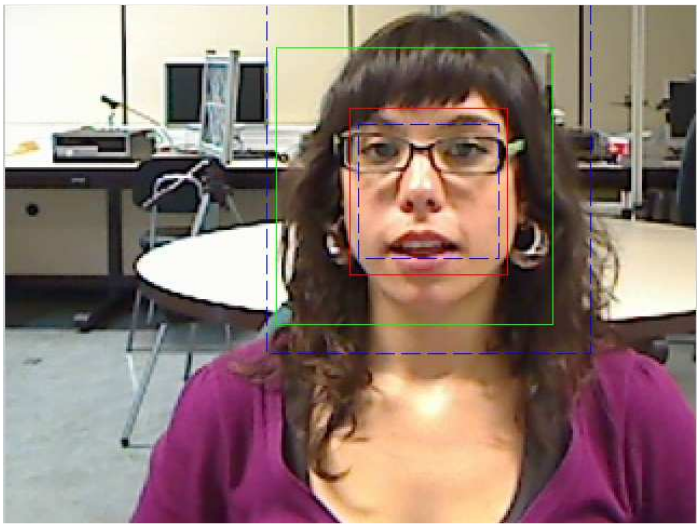


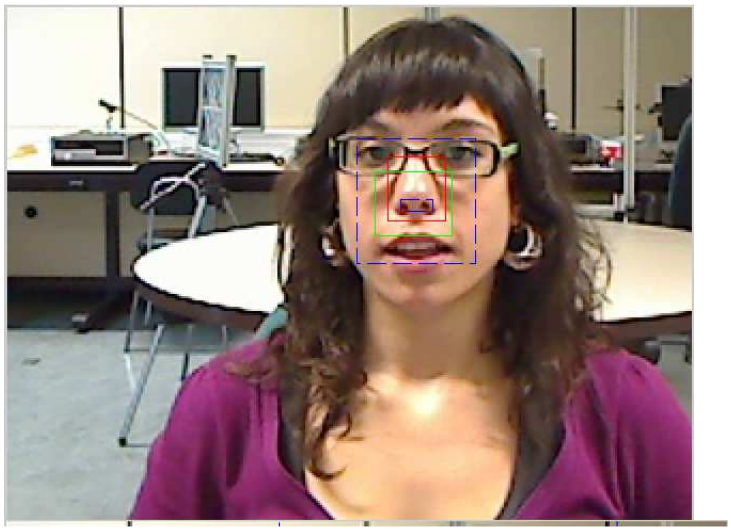
Figura. 4.10. TP. Detector mixto de cara, nariz y boca. Se detecta una nariz y dos bocas, la nariz y una boca pertenecen a los márgenes.



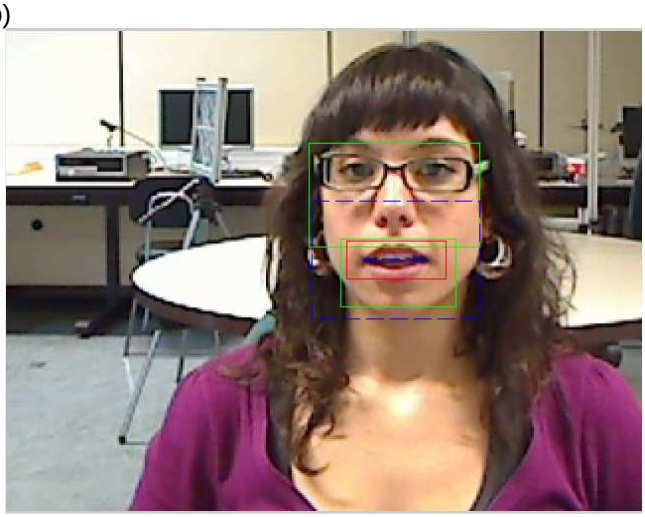
(a)



(b)



(c)



(d)

Figura. 4.11. a) zoom de figura 4.10 para mejor visualización de los resultados. b), c) y d) muestran las detecciones, y sus márgenes correspondientes, de la figura 4.10 por separado.

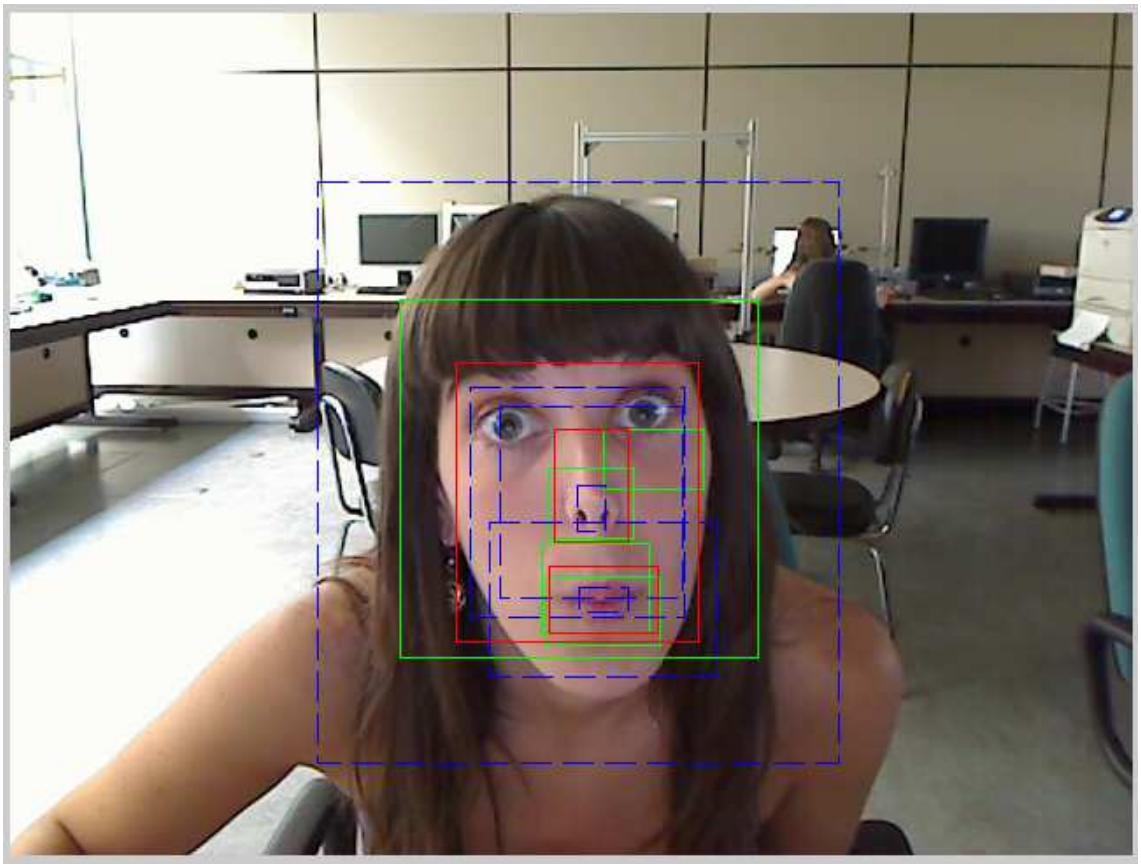


Figura. 4.12. TP. Detector mixto de cara, nariz y boca. Se detectan dos narices y dos bocas, una nariz y una boca quedan entre los márgenes.

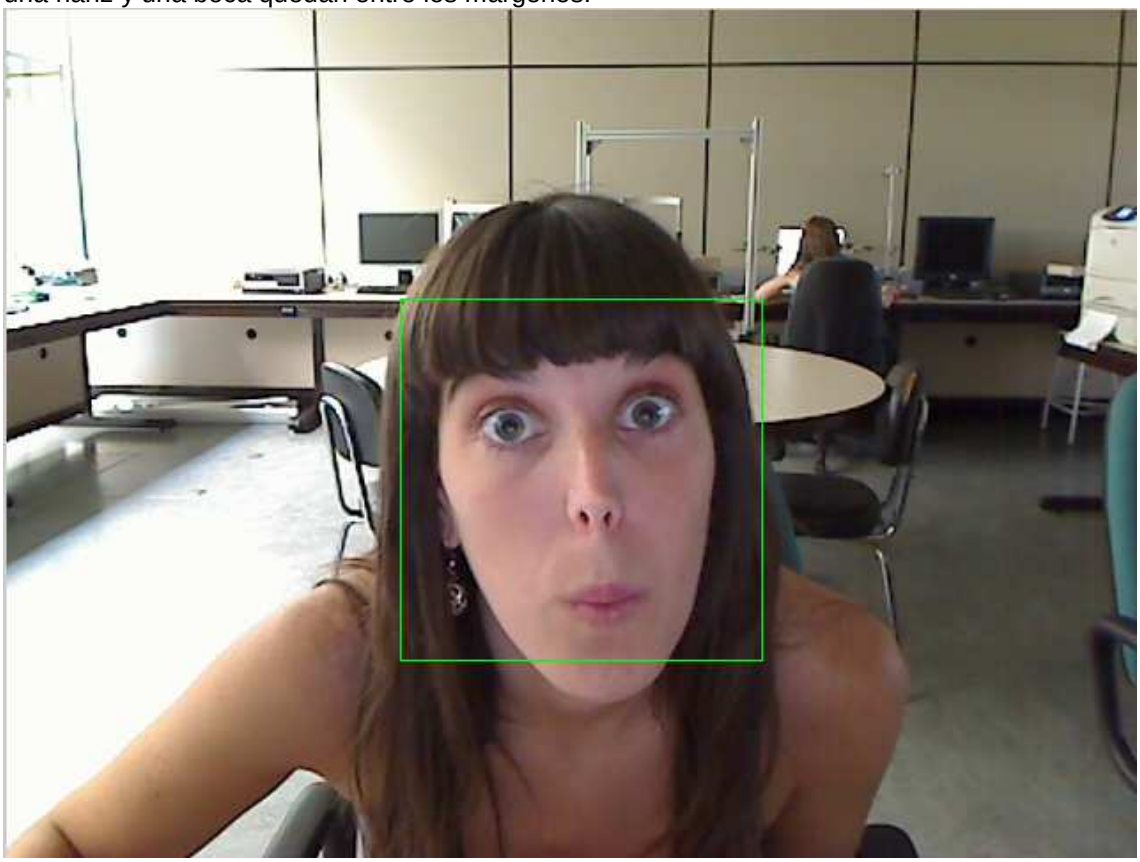


Figura. 4.13. Devolución en pantalla de la detección de cara utilizando un detector mixto de cara, nariz y boca.

Cuando *Test.m* finaliza, se devuelve en pantalla una tabla con los parámetros de estudio para cada test realizado. El titulo de la tabla son los xml empleados para la detección y se almacena en el directorio actual asignándole como nombre el propio título.

haarcascade-frontalface-alt2.xml
haarcascade-mcs-righteye.xml

	ACC (%)	FP	FN	TP	TN	TPR	SPC(TNR)	FPR
1	81.7925	55	396	2026	0	0.8365	0	1

Figura. 4.14. Tabla devuelta por Test.m en el caso de un detector mixto; cara y ojo derecho, utilizando los xml haarcascade_frontalface_alt2.xml y haarcascade_mcs_righteye.xml.

4.2.3 Funciones en C++

Debido a que la función principal de detección se encontraba escrita en lenguaje C++, se procedió a la compilación de la misma en formato MEX (MATLAB Executable File) de tal manera que pudiera ser embebida dentro de la función FaceDetection.m. Las funciones principales escritas en C++ son las siguientes:

4.2.3.1 CvHaarDetectObjects

La función cvHaarDetectObjects encuentra regiones rectangulares en la imagen que pueden contener los objetos buscados, la cascada ha sido entrenada para esas regiones, y los devuelve como una secuencia de rectángulos. La función analiza la imagen varias veces a diferentes escalas (ver cvSetImagesForHaarClassifierCascade). Cada vez que considere regiones superpuestas en la imagen se aplican los clasificadores a las regiones con cvRunHaarClassifierCascade. También puede aplicar un poco razonamiento heurístico para reducir el número de regiones analizadas, tales como poda Canny. Después del procesamiento y la reagrupación de los rectángulos candidatos (regiones que pasan la cascada del clasificador), se agrupan y devuelven como una secuencia de rectángulos promedio de cada grupo lo suficientemente grande

4.2.3.2 cvSetImagesForHaarClassifierCascade

La función cvSetImagesForHaarClassifierCascade asigna imágenes y/o ventanas de escala al clasificador de cascada oculta. La función se utiliza para preparar la cascada para la detección de objetos del tamaño concreto en la imagen concreta. La función se llama internamente por cvHaarDetectObjects, pero puede ser llamado por el usuario si es necesario en el uso de cvRunHaarClassifierCascade

4.2.3.3 cvRunHaarClassifierCascade

La función cvRunHaarHaarClassifierCascade ejecuta un clasificador Haar en cascada en una única localización de la imagen. Antes de utilizar esta función las imágenes integral y la escala apropiada (tamaño de la ventana) deben establecerse mediante cvSetImagesForHaarClassifierCascade. La función devuelve un valor positivo si los rectángulos analizados (candidatos) han superado todas las etapas del clasificador y devuelve cero o un valor negativo si no las superan.

4.3 DATOS DE PRUEBA

4.3.1 Base de datos de imágenes

4.3.1.1 Características

Para testear el algoritmo de detección de caras, es necesaria una base de datos de imágenes que contengan caras reales. Las múltiples bases de datos de caras ya existentes se pueden clasificar de acuerdo a sus propósitos, como por ejemplo las utilizadas para tecnología de detección facial o las utilizadas para la detección de la expresión facial. Debido a las características de este trabajo las bases de datos más adecuadas para realizar el testeo son las utilizadas para detección facial. Tras la realización de diferentes pruebas con distintas bases de datos se concluyó que lo más adecuado era la construcción de una base de datos de caras (BBDD) propia.

La base de datos utilizada en este proyecto está formada por imágenes realizadas con la cámara Logitech Webcam Pro 9000 y unificadas con las imágenes de la base de datos ya existente BioID-FaceDatabase-V1.2 [39]. Incluye un total de 2465 imágenes.

La ya mencionada BBDD BioID consta de 1511 imágenes en escala de grises con una resolución de 384 × 286 píxeles. Cada imagen muestra la vista frontal de la cara de 23 personas diferentes. La colección de imágenes de evaluación ofrece una amplia variedad de iluminación, fondo y tamaño de las caras.

Las imágenes tomadas con la Webcam suman un total de 954 imágenes de 53 sujetos diferentes. Se realizaron 18 fotografías por modelo, conteniendo una única cara sobre fondos heterogéneos. Cada imagen fue grabada en formato JPG con múltiples resoluciones (distinto tamaño): 640 × 480, 800 × 600, 965 × 686, 1018 × 619, 1201 × 901 y 1280 × 800. Durante la grabación de las imágenes se hizo especial énfasis en recrear las condiciones del "mundo real", por lo que las imágenes no sólo tienen diferente resolución, sino que también hay variación en la iluminación, la posición, los gestos o los tamaños de las caras. Además, se incluyeron variables como la toma de la foto en escala de grises o en color, la presencia o no de componentes estructurales (barba, gafas, pelo…) o la distancia a la que está tomada la imagen.

Todas las fotografías están etiquetadas en base a un nombre haciendo referencia a cada sujeto. Además en un archivo de texto están anotadas las posiciones de la cara, las características faciales (ojo izquierdo, derecho, nariz, boca, ambos ojos), las pupilas, la punta de la nariz y los extremos de la boca.

Figura. 4.15. Ejemplos de las imágenes tomadas por la Webcam.

Figura. 4.16. Ejemplos de imágenes de la base de datos BioID.

4.3.1.2 Tipos de imágenes

La base de datos de imágenes faciales debe incluir muchas variaciones por cada sujeto puesto que se utiliza para evaluar la validez y la precisión de los algoritmos de detección facial. Las diversas variaciones que deben estar incluidas en la base de datos se pueden dividir entre intrínsecas o extrínsecas. Las variables intrínsecas están relacionadas con la expresión facial, la ropa, la oclusión facial, el pelo, etcétera. Por su parte las variables extrínsecas incluyen características como la alineación, el fondo, el ajuste de la cámara, la iluminación, el dispositivo de imagen, la pose o la escala. Dentro de las numerosas bases de datos de caras ya existentes se pueden encontrar muchas variaciones de entre las que destacan la iluminación, la postura, la expresión, el cabello o la presencia o no de gafas. La base de datos generada con objeto de este estudio concreto incluirá tres variaciones intrínsecas (expresión, distancia y componentes estructurales) y cuatro variaciones extrínsecas (iluminación, pose, color y tamaño de la imagen).

4.3.1.2.1 Expresión

En la base de datos hay expresiones faciales de todo tipo ya que los modelos en cuestión tuvieron total libertad para cambiar la expresión de su cara en el momento de la toma de fotografías. La Figura 4.17 muestra algunos ejemplos.

Figura. 4.17. Ejemplos de imágenes del sujeto con diferentes expresiones.

4.3.1.2.2 Distancia

El tamaño de la cara en las imágenes de la base de datos varía ya que los sujetos de análisis fueron colocados a diferentes distancias de la cámara.

Figura. 4.18. Ejemplos de imágenes del sujeto a distintas distancias.

4.3.1.2.3 Componentes estructurales

Las componentes estructurales también están presentes en la base de datos. Como ejemplo está la existencia de barba, bigote, gafas o pelo, la largura de éste…

Figura. 4.19. Ejemplos de imágenes de sujetos con diversas componentes estructurales.

4.3.1.2.4 Iluminación

Las imágenes que componen la BBDD están tomadas en diversas condiciones de iluminación y sin seguir un patrón determinado.

Figura. 4.20. Ejemplos de imágenes del sujeto con múltiples condiciones de iluminación.

4.3.1.2.5 Pose

Al igual que en la expresión y en la distancia los sujetos tuvieron libertad total de movimientos y las poses son diferentes en cada imagen.

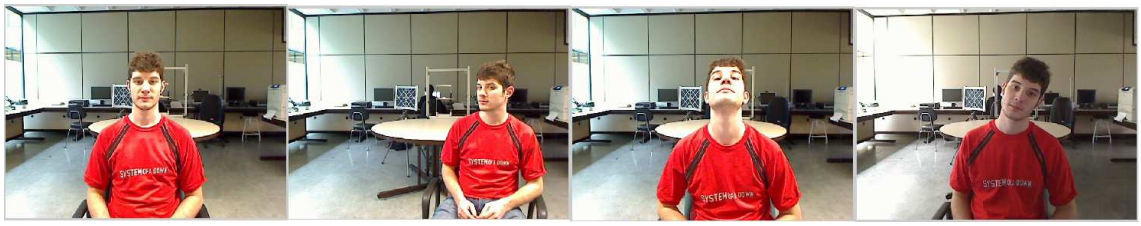


Figura. 4.21. Ejemplos de imágenes del sujeto en diversas poses.

4.3.1.2.6 Color

La base de datos utilizada en este estudio contiene imágenes tanto en color como en escala de grises. Las imágenes provenientes de la base de datos BioID están en escala de grises mientras que las tomadas manualmente con la cámara web combinan ambos formatos.

Figura. 4.22. Ejemplos de imágenes en color y en escala de grises. Las dos últimas imágenes pertenecen a la BBDD BioID.

4.3.1.2.7 Tamaño de la imagen

Como se ha mencionado anteriormente los distintos tamaños de las imágenes que componen la BBDD son: 384 × 286 (imágenes de la base de datos BioID), 640 × 480, 800 × 600, 965 × 686, 1018 × 619, 1201 × 901 y 1280 × 800. El funcionamiento del algoritmo de detección de caras debe ser evaluado utilizando imágenes de diferentes resoluciones

Figura. 4.23. Ejemplos de imágenes en las distintas resoluciones. (a) 384 × 286, (b) 640 × 480, (c) 800 × 600, (d) 965 × 686, (e) 1018 × 619, (f) 1201 × 901, (g) 1280 × 800.

4.3.2 Datos de etiquetado

La base de datos está ordenada en tres carpetas tal como se puede observar en la Figura 4.24. La carpeta “images” contiene todas las imágenes en formato jpg mientras que las otras dos almacenan archivos de texto con las posiciones de cada imagen etiquetadas adecuadamente, estas posiciones quedan descritas en el punto denominado *LabelsFace.* La primera carpeta alberga 76 subcarpetas asignadas una a cada sujeto e incluyendo las imágenes de un mismo modelo. Las otras dos carpetas contienen 76 archivos de texto cada una, es decir, uno por individuo. La carpeta “LabelsFace” contiene los archivos de texto con las posiciones, en pixeles, de la cara, ojos, nariz y boca; mientras que la carpeta “LabelsPoints” contiene las posiciones de las pupilas, punta de la nariz y extremos de la boca.

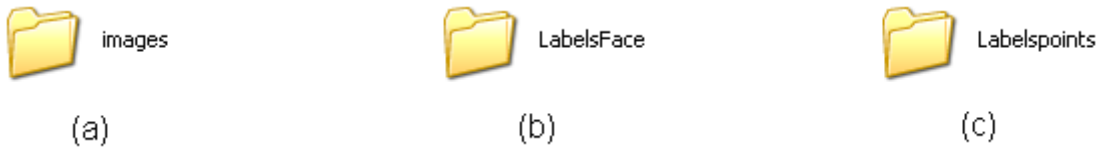


Figura. 4.24. Ordenación de la BBDD. (a) Carpeta contenedora de las imágenes de la BBDD, (b) y (c) carpetas contenedoras de los archivos de texto con las posiciones.

Cada una de las imágenes incluidas en las subcarpetas, denominadas “fxxx”, de la carpeta “images” ha sido nombrada siguiendo el estándar "f*xxx_yyy*.jpg", donde *xxx* corresponde al número de sujeto (y por tanto al número de la subcarpeta) e *yyy* al número de imagen de un mismo sujeto. Por ejemplo, la imagen 2 del sujeto número 13 sería f13_02.jpg.

Los archivos de texto han sido nombrados siguiendo el modelo “LabelsFacef*xxx*.txt” o “Labelspointsf*xxx*.txt” dependiendo de la carpeta a la que pertenezcan siendo xxx el número de sujeto. Por ejemplo, el archivo de texto que contiene las posiciones de la cara, ojos, nariz y boca del sujeto número 13, se denomina LabelsFacef13.txt.

4.3.2.1 Etiquetas de caras y características faciales

Un archivo txt LabelsFace contiene una matriz de números con tantas filas como imágenes incluye la carpeta a la que se refiere y 24 columnas. Cada fila son las posiciones de la cara y de las características faciales para cada imagen (la cuarta fila de LabelsFacef13.txt son las posiciones de la imagen 4 del sujeto 13). Estas posiciones quedan descritas con dos coordenadas ((x_1, y_1), (x_2, y_2)) por cada elemento: cara, ojo izquierdo, ojo derecho, nariz, boca y dos ojos en global (ver Figura 4.25 y 4.26). Estos dos puntos definen un rectángulo que contiene la cara o característica facial determinada. (x_1, y_1) es la esquina superior izquierda del rectángulo y (x_2, y_2) la esquina inferior derecha (ver Figura 4.26). En el caso de que la imagen no contenta una cara o una determinada característica facial, las coordenadas numéricas correspondientes se sustituyen por NaN (not a number). Hay que tener en cuenta que cuando se etiqueta el ojo izquierdo, realmente se etiqueta el ojo izquierdo de la persona, y éste aparece al lado derecho en la imagen.

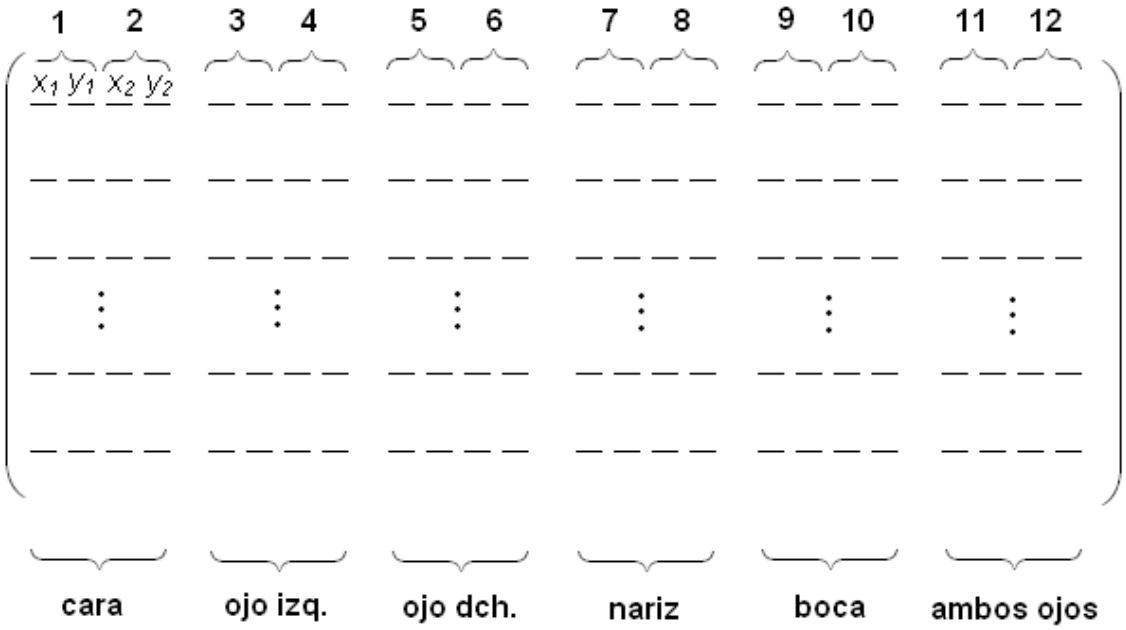


Figura. 4.25. Matriz del archivo txt que contiene las posiciones de LabelsFace.

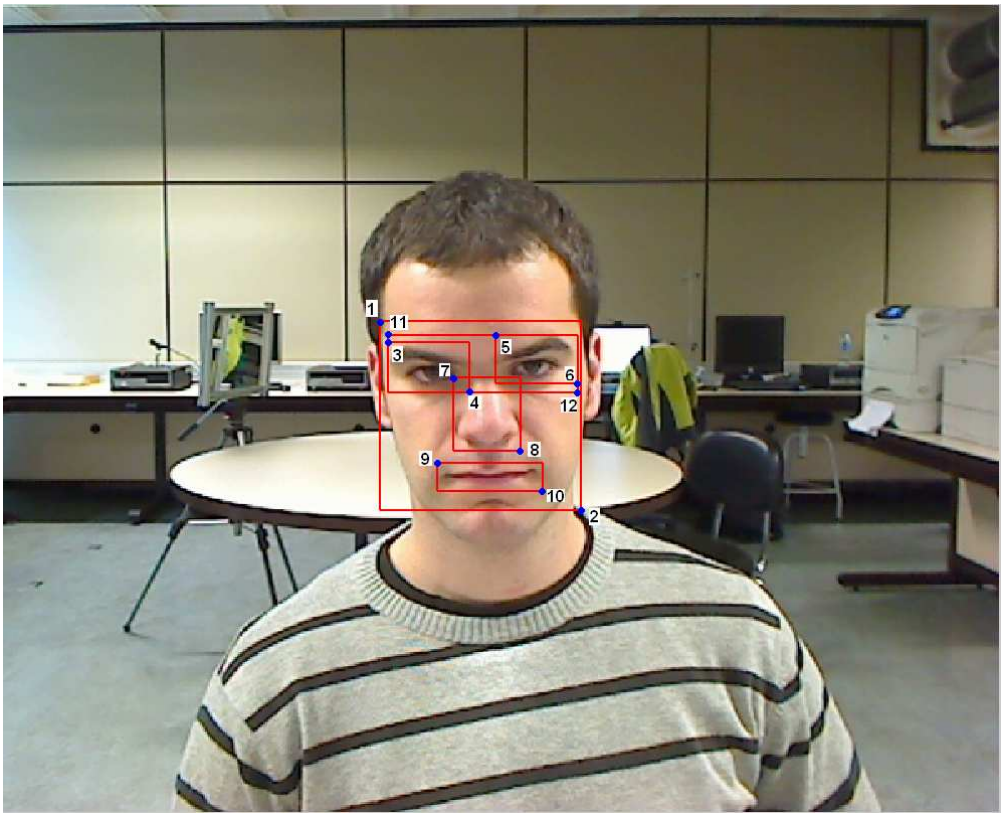


Figura. 4.26. Ejemplo del etiquetado LabelsFace de las características faciales de una imagen de la base de datos. Los puntos 1 y 2 son las coordenadas (x_1, y_1) y (x_2, y_2) que conforman el rectángulo contenedor de la cara.

El etiquetado de todas las imágenes ha sido realizado a mano utilizando una herramienta del programa MATLAB. En el caso de la base de datos BioID, las imágenes estaban etiquetadas sólo para los valores de Labelspoints (ver Figura 4.27). A partir de éstas etiquetas se calcularon, mediante el uso del programa MATLAB (script *etiquetar.m* y *CambiarNombre.m* en ANEXO 1), las posiciones para LabelsFace (ver Figura 4.27).

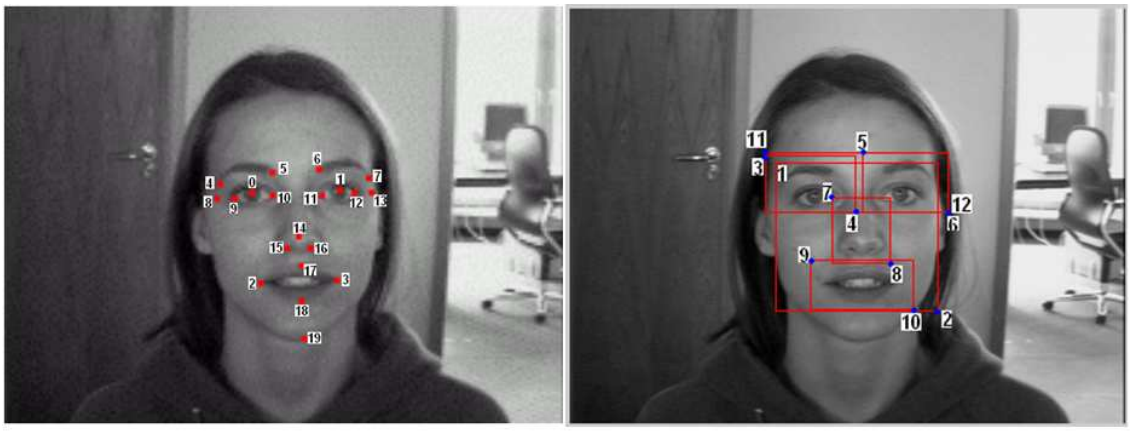


Figura. 4.27. (a) Ejemplo del etiquetado de una imagen de la base de datos BioID, (b) Etiquetado final una imagen de la base de datos BioID.

4.3.2.2 Etiquetas de puntos representativos

Un archivo txt Labelspoints contiene una matriz de números con tantas filas como imágenes incluye la carpeta a la que se refiere y 28 columnas. Cada fila son las posiciones de las cejas, los ojos, la nariz y la boca. La cuarta fila de Labelspointsf13.txt serían las posiciones de la imagen 4 del sujeto 13. Estas posiciones quedan descritas con una coordenada *(x,y)* por elemento (ver Figuras 4.28, 4.29 y 4.30). De nuevo las etiquetas del ojo izquierdo se refieren al ojo izquierdo del sujeto apareciendo en la parte derecha de la imagen.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Ceja								**Ojo**												**nariz**				**boca**			
izda				dcha				izdo						dcho						izda		dcha		izda		dcha	
exterior		interior		interior		exterior		exterior		pupila		interior		interior		pupila		exterior									
x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y	x	y

Figura. 4.28. Matriz del archivo txt que contiene las posiciones de Labelspoints.

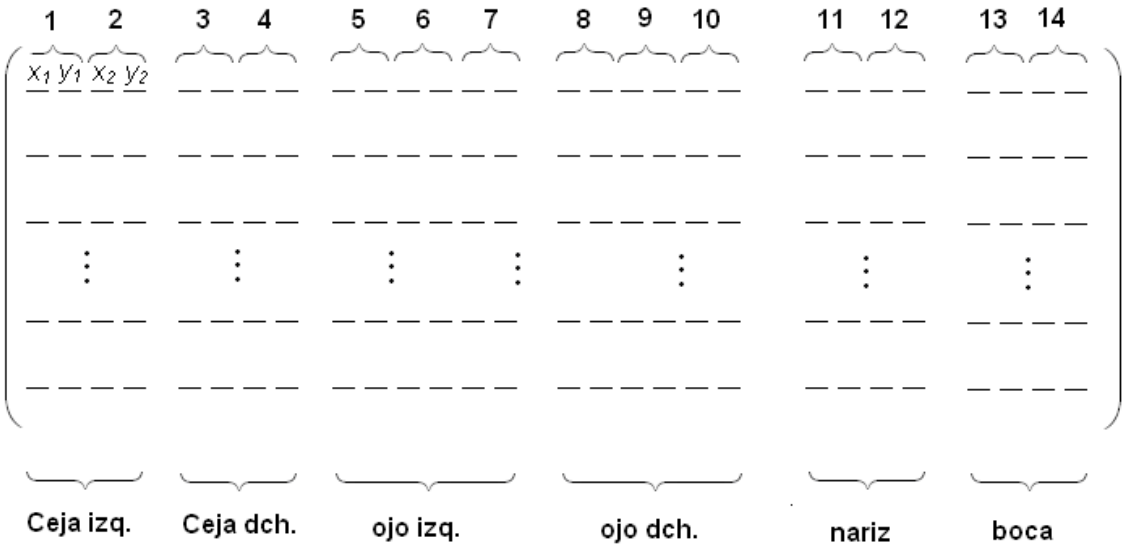


Figura. 4.29. Matriz del archivo txt que contiene las posiciones de Labelspoints

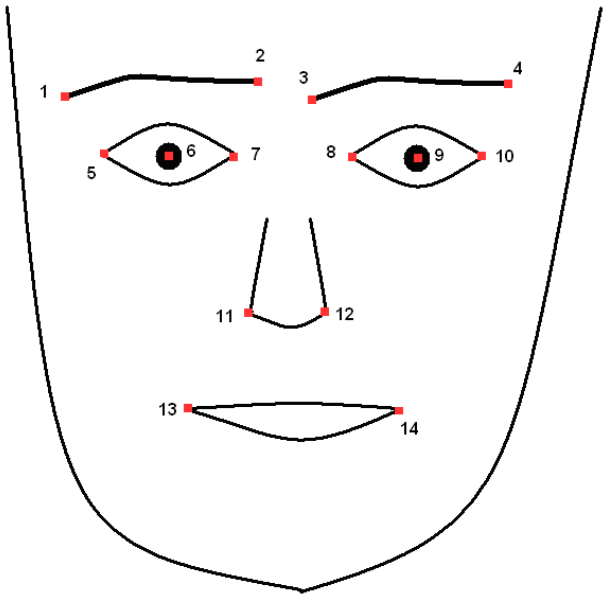


Figura. 4.30. Ejemplo del etiquetado Labelspoints en una cara.

En este caso se seleccionaron los puntos deseados. Por ejemplo la coordenada 13 correspondiente a la parte izquierda de la boca es el punto 2 en la base de datos BioID (ver Figura 4.31).

(a) (b)

Figura. 4.31. (a) Ejemplo del etiquetado de una imagen de la base de datos BioID, (b) Etiquetado final una imagen de la base de datos BioID.

4.4 ETAPAS DE LA IMPLEMENTACIÓN

En la tabla inferior se adjunta la descripción de las sucesivas etapas en las que consistió la implementación.

Etapa	Duración temporal	Tarea
Documentación	Febrero 2010- Abril 2010	Recopilación de información
BBDD	Abril 2010 - Mayo 2010	Creación de la base de datos
Toma de contacto; Matlab + C	Julio 2010	Interpretación de los archivos xml implementados en C
Adquisición conocimientos C++	Agosto 2010	Adquirir conocimientos de C para modificar algoritmo implementado en C++
Matlab	Septiembre 2010 – Noviembre 2010	Implementación en Matlab
Simulaciones	Diciembre 2010 – Enero 2010	

Tabla 4.1. Etapas de la implementación

5 EVALUACIÓN Y RESULTADOS

En este apartado se exponen los resultados obtenidos a partir de las diversas simulaciones realizadas utilizando los diferentes detectores de caras y características faciales.

El objetivo tras la realización de estas simulaciones es saber que combinación de archivos xml y que parámetros optimizan la detección de caras en imágenes o secuencias de video.

Para evaluar el funcionamiento de los detectores se describen los resultados obtenidos al aplicarlos sobre la base de datos descrita en el apartado 4.3 de esta memoria.

5.1 CLASIFICADORES PROPUESTOS

1. Detección de cara utilizando el archivo haarcascade_frontalface_alt.xml.

2. Detección de cara utilizando el archivo haarcascade_frontalface_alt2.xml.

3. Detección de cara utilizando el archivo haarcascade_frontalface_alt_tree.xml.

4. Detección de cara utilizando el archivo haarcascade_frontalface_default.xml.

5. Detección de cara-ojo derecho utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_righteye_2splits.xml.

6. Detección de cara-ojo derecho utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_righteye.xml.

7. Detección de cara-ojo izquierdo utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_lefteye_2splits.xml.

8. Detección de cara-ojo izquierdo utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_lefteye.xml.

9. Detección cara-nariz utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_nose.xml.

10. Detección cara-boca utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_mouth.xml.

11. Detección cara-ambos ojos utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_eyepair_big.xml.

12. Detección cara-ambos ojos utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_eyepair_small.xml.

13. Detección de cara-ojoizquierdo-ojoderecho utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml. y haarcascade_righteye_2splits.xml.

14. Detección de cara-ojo izquierdo-ojo derecho utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml. y haarcascade_mcs_righteye.xml.

15. Detección de cara-ojoizquierdo-nariz utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml., y haarcascade_mcs_nose.xml.

16. Detección de cara-ojoizquierdo-nariz utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., y haarcascade_mcs_nose.xml.

17. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml. y haarcascade_mcs_nose.xml.

18. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml. y haarcascade_mcs_mouth.xml.

19. Detección de cara-ojoizquierdo-ojoderecho-nariz-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

20. Detección de cara-ojoizquierdo-nariz-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_lefteye_2splits.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

21. Detección de cara-ojoizquierdo-nariz-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

22. Detección cara-nariz-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

23. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml. y haarcascade_mcs_nose.xml.

24. Detección de cara-ojoizquierdo-ojoderecho-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml. y haarcascade_mcs_mouth.xml.

25. Detección de cara-ojoizquierdo-ojoderecho-nariz-boca utilizando los archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml, haarcascade_mcs_nose.xml. y haarcascade_mcs_mouth.xml.

26. Detección de cara-ojo derecho utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_righteye_2splits.xml.

27. Detección de cara-ojo derecho utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_righteye.xml.

28. Detección de cara-ojo izquierdo utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_lefteye_2splits.xml.

29. Detección de cara-ojo izquierdo utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_lefteye.xml.

30. Detección cara-nariz utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_nose.xml.

31. Detección cara-boca utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_mouth.xml.

32. Detección cara-ambos ojos utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_eyepair_big.xml.

33. Detección cara-ambos ojos utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_eyepair_small.xml.

34. Detección de cara-ojoizquierdo-ojoderecho utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml. y haarcascade_righteye_2splits.xml.

35. Detección de cara-ojo izquierdo-ojo derecho utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml. y haarcascade_mcs_righteye.xml.

36. Detección de cara-ojoizquierdo-nariz utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml., y haarcascade_mcs_nose.xml.

37. Detección de cara-ojoizquierdo-nariz utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml., y haarcascade_mcs_nose.xml.

38. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml. y haarcascade_mcs_nose.xml.

39. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml. y haarcascade_mcs_mouth.xml.

40. Detección de cara-ojoizquierdo-ojoderecho-nariz-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml., haarcascade_righteye_2splits.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

41. Detección de cara-ojoizquierdo-nariz-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_lefteye_2splits.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

42. Detección de cara-ojoizquierdo-nariz-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

43. Detección cara-nariz-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_nose.xml y haarcascade_mcs_mouth.xml.

44. Detección de cara-ojoizquierdo-ojoderecho-nariz utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml. y haarcascade_mcs_nose.xml.

45. Detección de cara-ojoizquierdo-ojoderecho-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml. y haarcascade_mcs_mouth.xml.

46. Detección de cara-ojoizquierdo-ojoderecho-nariz-boca utilizando los archivos haarcascade_frontalface_alt2.xml, haarcascade_mcs_lefteye.xml., haarcascade_mcs_righteye.xml, haarcascade_mcs_nose.xml. y haarcascade_mcs_mouth.xml.

5.2 PARÁMETROS DE CAMBIO

5.2.1 Parámetros en cvod200uint_v02.cpp

Flag → 0/Canny_prunning
Ecualización → si/no

5.2.2 Parámetros de entrada de cv200uint_v02.cpp

Escala →1/2
Tamaño mínimo de la ventana de búsqueda → detección de cara – detección de características.

En las primeras pruebas realizadas se decidió fijar el parámetro flag ya que sin falta de realizar todas las simulaciones quedaba claro que los resultados obtenidos serían mejores.

Flag → Canny_prunning

5.3 CRITERIOS DE EVALUACIÓN

La evaluación está fundamentada en una serie de parámetros definidos a continuación. Se hace especial hincapié en el parámetro ACC, exactitud.

5.3.1 Definición de parámetros de test

Un contraste de hipótesis es un método de toma de decisiones a partir de datos experimentales.

Mediante esta teoría, se aborda el problema estadístico considerando una hipótesis determinada H_0 y una hipótesis alternativa H_1, y se intenta concluir cuál de las dos

es la hipótesis verdadera, tras aplicar el problema estadístico a un cierto número de experimentos.

En un contraste de hipótesis, pueden extraerse dos tipos de conclusiones incorrectas. La hipótesis es rechazada cuando es cierta (error de tipo I), o aceptar la hipótesis cuando es falsa (error de tipo II).

	H_0 **es cierta**	H_1 **es cierta**
Aceptar H_0	No hay error	Error de tipo II
Aceptar H_1	Error de tipo I	No hay error

Tabla 5.1. Tabla de tipos de error.

El error de tipo I, también conocido como "falso positivo" es el error de rechazar una hipótesis nula cuando en realidad es cierta. Por ejemplo, un tribunal decide que una persona es culpable de un crimen que él o ella en realidad no cometió.

El Error de Tipo II puede llamarse también "falso negativo" y es el error de no rechazar la hipótesis nula, dado que la hipótesis alternativa es realmente cierta, por ejemplo, un tribunal decide que una persona no es culpable de un crimen que él o ella ha cometido.

Dentro del contexto de este proyecto, un **falso positivo** es aceptar que se ha detectado un objeto cuando en realidad el objeto no está en la imagen y un **falso negativo** es no detectar el objeto cuando éste si pertenece a la imagen.

Cuando no se cometen errores también existen dos alternativas, “positivo verdadero” o “**True Positive**” detectar el objeto cuando éste pertenece a la imagen y “negativo verdadero” o “**True Negative**”, no detectar el objeto cuando no está en la imagen.

	Objeto pertenece a la imagen	**Objeto no pertenece a la imagen**
Detectar objeto	True Positive (TP)	Falso Positivo (FP)
No detectar objeto	Falso Negativo (FN)	True Negative (TN)

Tabla 5.2. Tabla de confusión.

Existen dos medidas estrechamente relacionadas con los conceptos de falso positivo y falso negativo, sensitivity (sensibilidad) o **True positive rate** (TPR) y specificity (especificidad) o **True Negative Rate** (SPC o TNR).

La sensibilidad o TPR (también llamado índice de repetición en algunos campos) mide la proporción de positivos reales que son correctamente identificados como tales (por ejemplo, el porcentaje de enfermos que están correctamente identificados con la enfermedad). Hay un parámetro asociado a la sensibilidad, **False positive rate** (FPR), éste define el número de resultados positivos incorrectos que se producen en la prueba.

$$SPC = TNR = \frac{TrueNegatives}{TrueNegatives + FalsePositives} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FalsePositives}{FalsePositives + TrueNegatives} = \frac{FP}{FP + TN} = 1 - SPC$$

La especificidad o TNR mide la proporción de negativos que se han identificado correctamente (por ejemplo, el porcentaje de personas sanas que son correctamente identificados por no tener la enfermedad).

$$TPR = \frac{TruePositives}{TruePositives + FalseNegatives} = \frac{TP}{TP + FN}$$

Una especificidad del 100% significa que el test reconoce todas los negativos reales, no hay positivos erróneamente etiquetados, por ejemplo, todas las personas sanas serán reconocidas como saludables. La especificidad por sí sola no nos dice como de bien reconoce el test los casos positivos también tenemos que conocer la sensibilidad de la prueba. Así, una sensibilidad del 100% significa que la prueba reconoce todos los positivos reales.

Una prueba con alta especificidad tiene un error de tipo I bajo, es decir tiene pocos falsos positivos y una prueba con una alta sensibilidad tiene un error de tipo II bajo, falsos negativos mínimos.

La proporción de verdaderos resultados (tanto positivos verdaderos como verdaderos negativos) en la muestra es la exactitud o **accuracy** (ACC). La exactitud es el grado de veracidad, es decir, la exactitud es cómo de cerca del valor real se encuentra el valor medido.

$$ACC = \frac{TruePositives + TrueNegatives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives} = \frac{TP + TN}{TP + FP + TN + FN}$$

Una exactitud del 100% significa que los valores medidos son exactamente los mismos que los valores dados.

La definición de estos parámetros queda concretada en la función implementada sobre matlab, *Test.m*, explicada en el punto 4.2.2.3.

5.4 RESULTADOS

Los mejores resultados son aquellos que maximizan la ACC. A continuación se muestra una tabla que recoge los resultados obtenidos. Todos ellos son valores para ACC en %. También se muestran algunas gráficas para facilitar la interpretación de los resultados.

Parámetros de cambio			Detectores														
Size ventana	Ecualizacion	Escala	alt	alt2	tree	default	5	6	7	8	9	10	11	12	13	14	
60-30	EQ	1	60,64	50,01	77,97	48,21	51,96	52,45	66,28	64,74	72,21	73,42	19,76	11,68	45,84	48,23	
60-30	EQ	2	76,66	73,08	78,94	71,36	18,05	12,12	16,3	13,75	22,1	15,21	2,71	1,05	12,98	10,1	
60-30	No EQ	1	86,79	84,27	86,11	78,46	58,53	62,79	72,9	69,49	80	75,53	24,58	13,07	53,87	58,21	
60-30	No EQ	2	81,27	80,92	80,34	75,72	15,9	10,5	14,27	12,29	19,18	16,86	7,34	4,81	13,87	14,83	
60-15	EQ	1	60,64	50,01	77,97	48,21	54,27	68,84	77,24	82,02	**90,34**	77,89	26,41	15,91	52,12	70,75	
60-15	EQ	2	76,66	73,08	78,94	71,36	49,65	49,85	51,23	65,27	75,17	75,05	21,98	10,62	39,67	48,76	
60-15	No EQ	1	86,79	84,27	86,11	78,46	50,22	65,27	72,61	78,53	86,77	81,25	31,42	22,43	46,69	61,17	
60-15	No EQ	2	81,27	80,92	80,34	75,72	56,06	60,12	58,05	70,7	76,1	77,32	23,57	11,31	48,72	58,78	

alt																
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
61,86	60,32	43,95	38,9	37,4	52,29	52,08	64,46	47,26	41,21	40,77	51,23	52,25	65,11	64,34	78,74	71,88
14,68	12,9	12,61	9,73	9,85	11,19	10,75	13,42	9,89	9,29	9,37	17,56	11,92	15,41	12,37	22,51	14,48
66,28	63,65	50,54	46	45,87	61,24	58,97	67,5	54,87	49,26	48,52	57,64	61,7	73,02	68,39	80,08	74,48
14,95	14,8	11,54	10,56	12,54	12,46	13,08	13,84	11,21	12,51	11,28	16,2	11,2	13,64	10,98	18,75	16,34
77,16	82,31	55,61	46,04	49,57	67,42	73,63	78,58	70,75	63,85	65,19	54,07	68,19	76,22	80,85	88,8	76,55
47,84	60,56	38,37	35,49	34,84	42,67	55,53	66,53	47,14	46,04	44,74	47,95	50,34	50,38	64,84	75,78	74,4
72,86	74,4	45,31	38,94	38,09	62,63	65,11	75,17	57,8	52,86	51,07	49,92	64,18	72,86	77,44	86,08	80,76
53,54	64,5	46,65	43,89	42,47	48,55	60,28	68,76	55,78	54,44	52,33	56,1	59,71	58,33	71,68	76,47	75,82

alt2														
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
17,36	10,08	45,31	48,19	60,64	60,68	45,31	38,54	37,12	52,12	51,86	63,48	46,98	40,85	40,87
2,14	1,02	11,88	9,49	13,75	12,09	11,15	8,72	8,64	12,66	10,1	12,61	9,49	8,55	8,6
23,77	12,68	53,46	57,36	66,32	64,01	51,83	45,87	45,35	57,11	55,05	66,93	54,16	48,86	48,49
6,52	3,83	13,36	13,57	13,42	14,61	13,61	10,31	12,65	12,28	12,67	13,2	10,85	11,67	11,18
24,12	11,84	52,41	69,57	76,51	80,97	55,21	46,77	48,92	65,76	72,21	76,71	70,02	63,36	64,09
21,46	12,84	38,82	49,24	47,42	61,62	37,93	33,18	32,77	40,85	55,78	66,16	47,58	45,23	44,17
28,67	19,87	46,24	60,48	72,79	74,13	47,21	38,24	37,82	59,84	63,45	74,86	57,53	52,13	50,99
23	12,24	48,51	58,49	53,34	65,92	46,33	43,08	41,7	47,54	60,28	68,19	55,65	53,67	52

Tabla 5.3. Resultados del testeo de todas las imágenes para cada combinación de parámetros y detectores. ACC en %.

ACC (%) > 90

85 < ACC (%) < 90

80 < ACC (%) < 85

70 < ACC (%) < 80

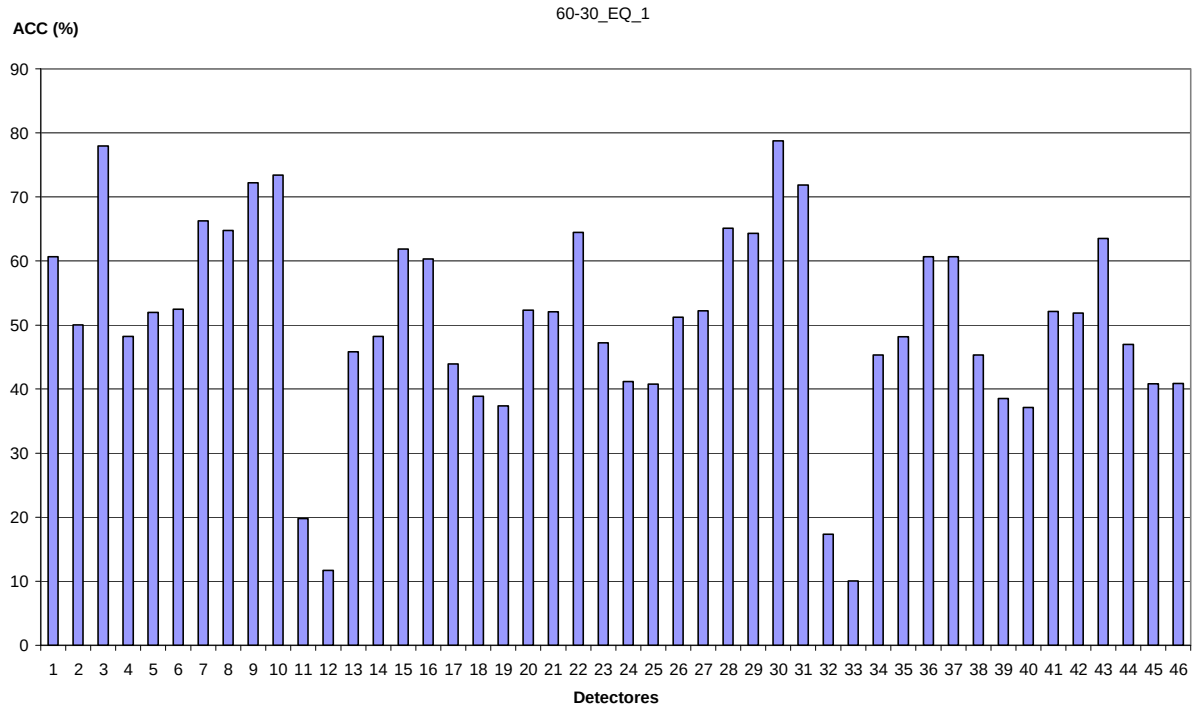


Figura 5.1. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-30, Ecualización y escala=1.

En esta gráfica se presentan los valores de ACC obtenidos para los detectores con tamaño de venta 60-30, ecualización y escala 1. Se observa que el mejor valor para ACC es para el detector 30, alcanzando casi un 80%. Los detectores 11, 12, 32 y 33, muestran unos valores de ACC claramente inferiores al resto, y por debajo del 20%. Únicamente 5 detectores quedan por encima del 70%, el detector 3, 9, 10, 30 y 31, pero ninguno de éstos supera el 80%.

Por facilitar la comprensión del párrafo anterior se vuelven a escribir a continuación los nombres de los detectores comentados en éste.

3. detección de cara utilizando el archivo haarcascade_frontalface_alt_tree.xml.,

9. detección cara-nariz con archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_nose.xml.

10. detección cara-boca con archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_mouth.xml.

11. detección de cara-ambosojos utilizando los archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_eyepair_big.xml

12. detección de cara-ambosojos con archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_eyepair_small.xml.

30. detección cara-nariz utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_nose.xml

31. detección cara-boca con archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_mouth.xml

32. detección de cara-ambos ojos con los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_eyepair_big.xml.

33. detección de cara-ambosojos con los archivos haarcascade_frontalface_alt.2xml y haarcascade_mcs_eyepair_small.xml.

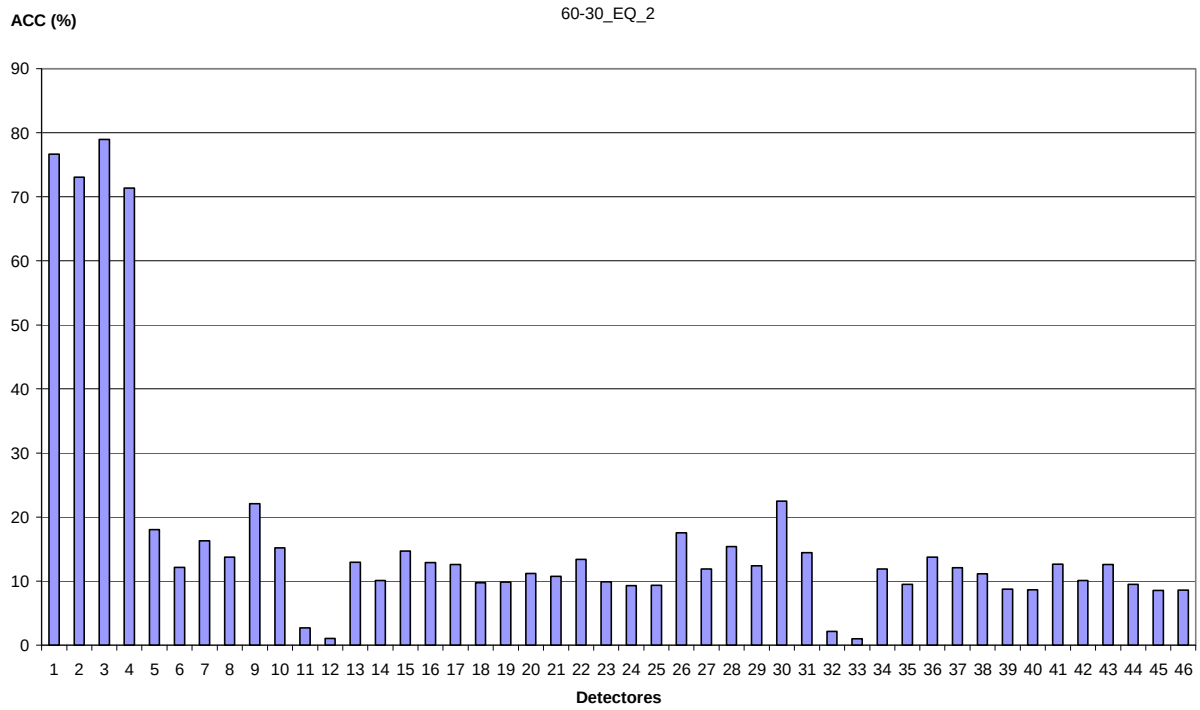


Figura 5.2. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-30, Ecualización y escala=2.

La gráfica anterior muestra el mal rendimiento de los detectores mixtos para un tamaño de ventana de 60-30, ecualización y escala 2. Los detectores simples superan los cuatro el 70% pero ninguno de ellos alcanza el 80%.

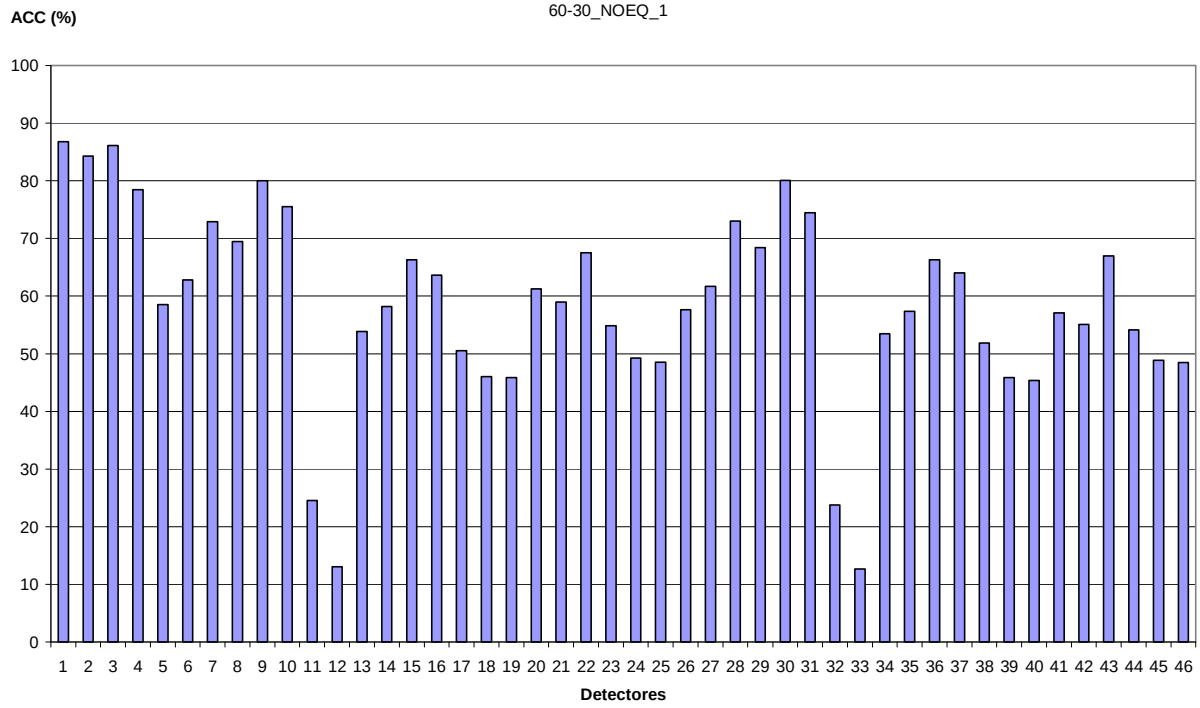


Figura 5.3. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-30, Sin ecualización y escala=1.

La figura 5.3 representa los valores para los detectores con tamaño de ventana 60-30, sin ecualización y con escala 1. De nuevo, los detectores 11, 12, 32 y 33 ofrecen muy malos resultados. Dos de los clasificadores, el 9 y el 30, rozan el 80 % y tres clasificadores, 1, 2 y 3, lo superan sin alcanzar el 90% (detección de cara con archivos haarcascade_frontalface_alt.xml., haarcascade_frontalface_alt2.xml. y haarcascade_frontalface_alt_tree.xml).

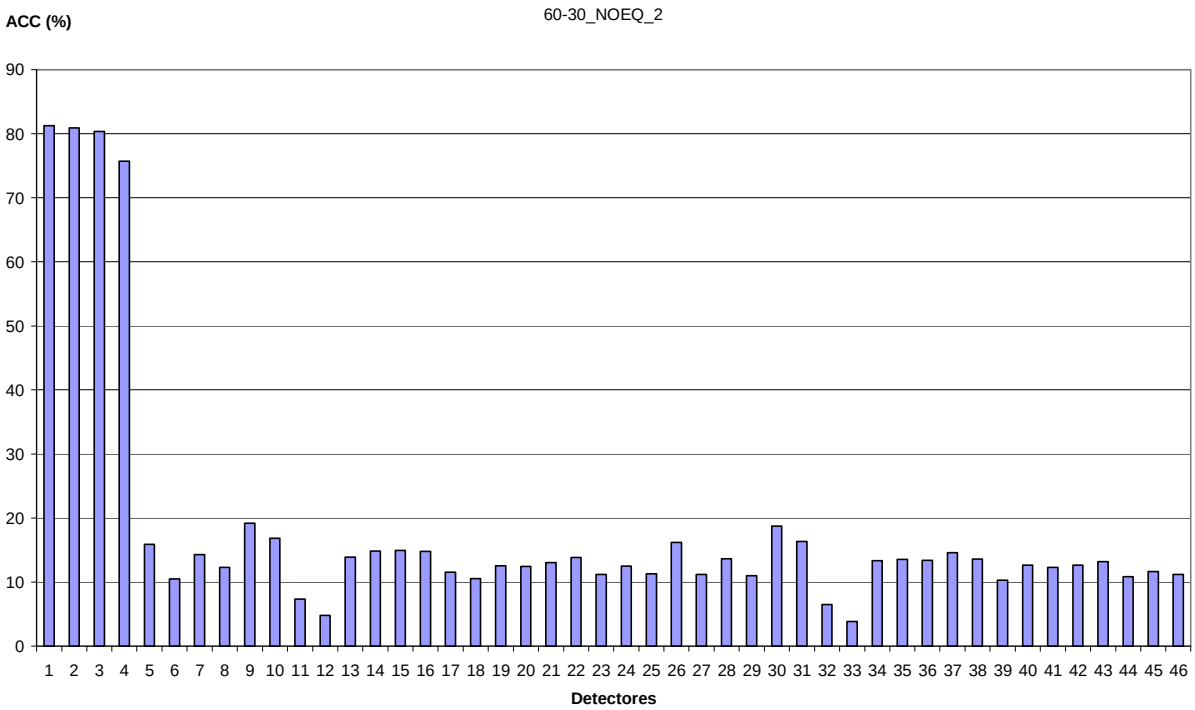


Figura 5.4. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-30, Sin ecualización y escala=2.

En la figura 5.4 observamos el mal rendimiento de los detectores mixtos para un tamaño de ventana de 60-30, sin ecualización y escala 2. Tres de los detectores simples superan ligeramente el 80%.

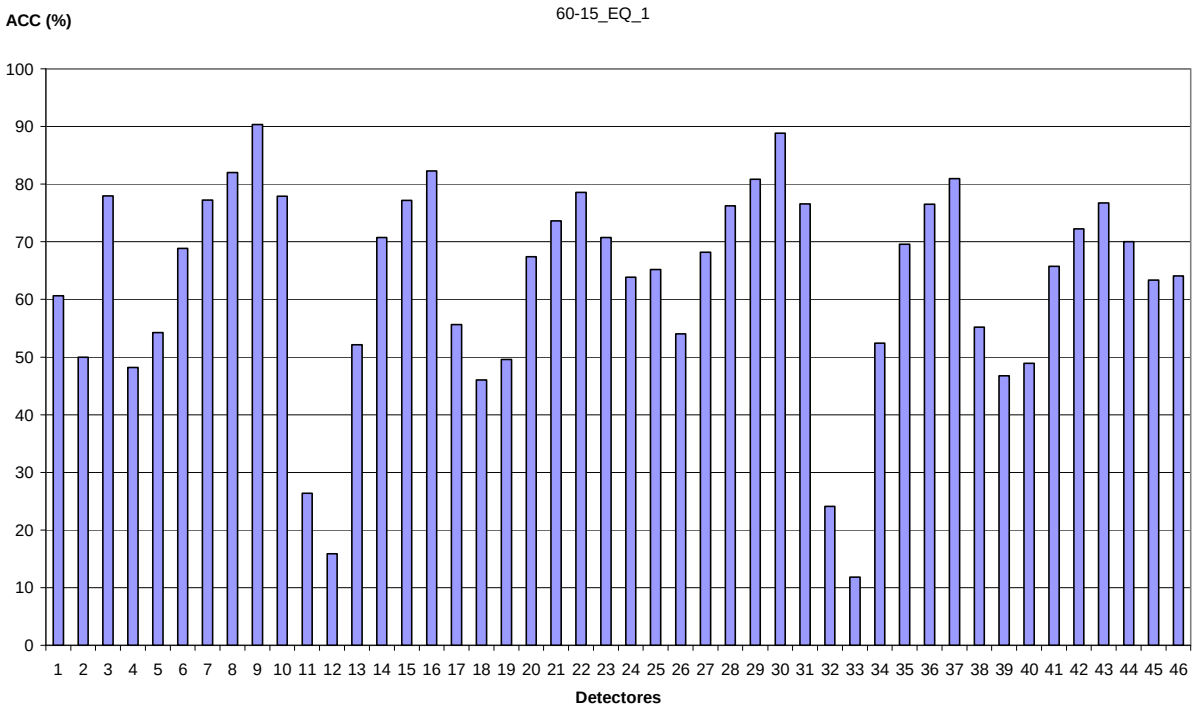


Figura 5.5. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-15, Ecualización y escala=1.

En la gráfica anterior observamos que para los valores de tamaño ventana 60-15, ecualización y escala 1, los malos resultados de los detectores 11, 12, 32 y 33 persisten. En este caso, 6 detectores superan el 80 %, el detector 8, detector 9, detector 16, detector 29, detector 30 y detector 37. Uno de estos 6 detectores, el detector 9, alcanza el 90%.

8. detección cara-ojoizq con archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_lefteye.xml.

9. detección cara-nariz utilizando archivos haarcascade_frontalface_alt.xml y haarcascade_mcs_nose.xml

16. detección cara-ojoIzq-nariz con archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., y haarcascade_mcs_nose.xml.

30. detección cara-nariz utilizando los archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_nose.xml.

29. detección cara-ojoizq con archivos haarcascade_frontalface_alt2.xml y haarcascade_mcs_lefteye.xml.)

37. detección cara-ojoIzq-nariz con archivos haarcascade_frontalface_alt.xml, haarcascade_mcs_lefteye.xml., y haarcascade_mcs_nose.xml.

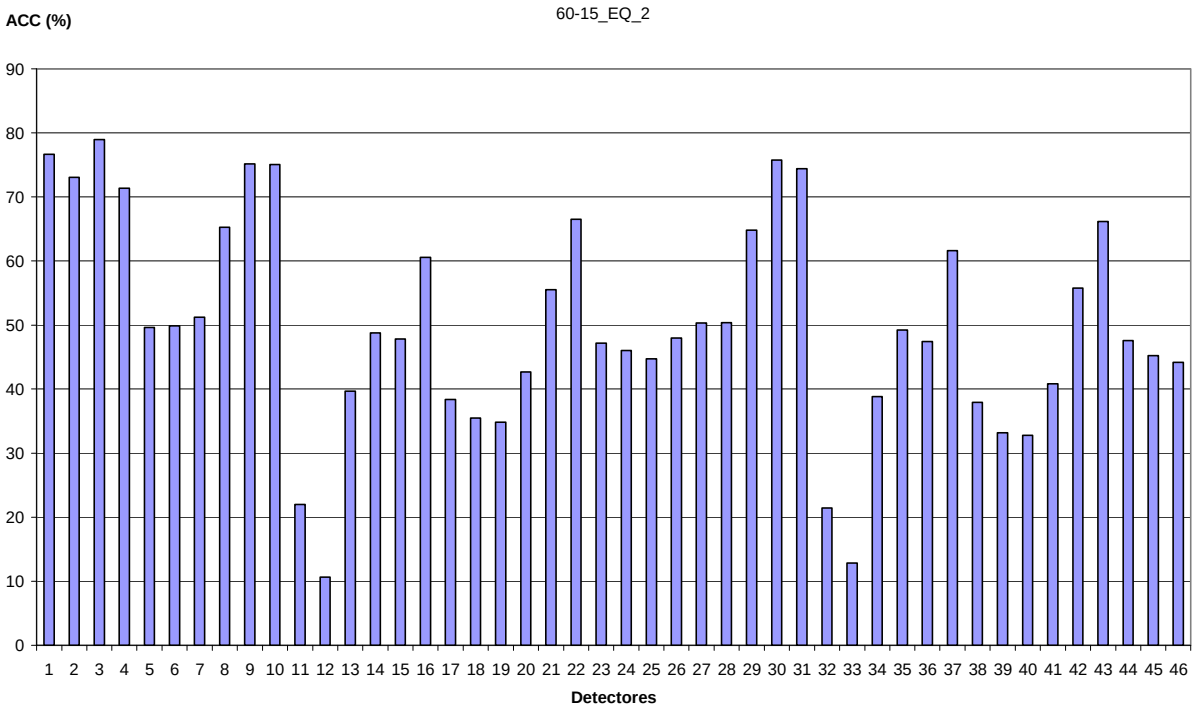


Figura 5.6. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-15, Ecualización y escala=2.

En el caso de tamaño de ventana 60-15, ecualización y escala 2, ningún detector alcanza el 80%, obteniéndose los mejores resultados, casi un 80%, para el detector simple de cara haarcascade_frontalface_alt_tree.

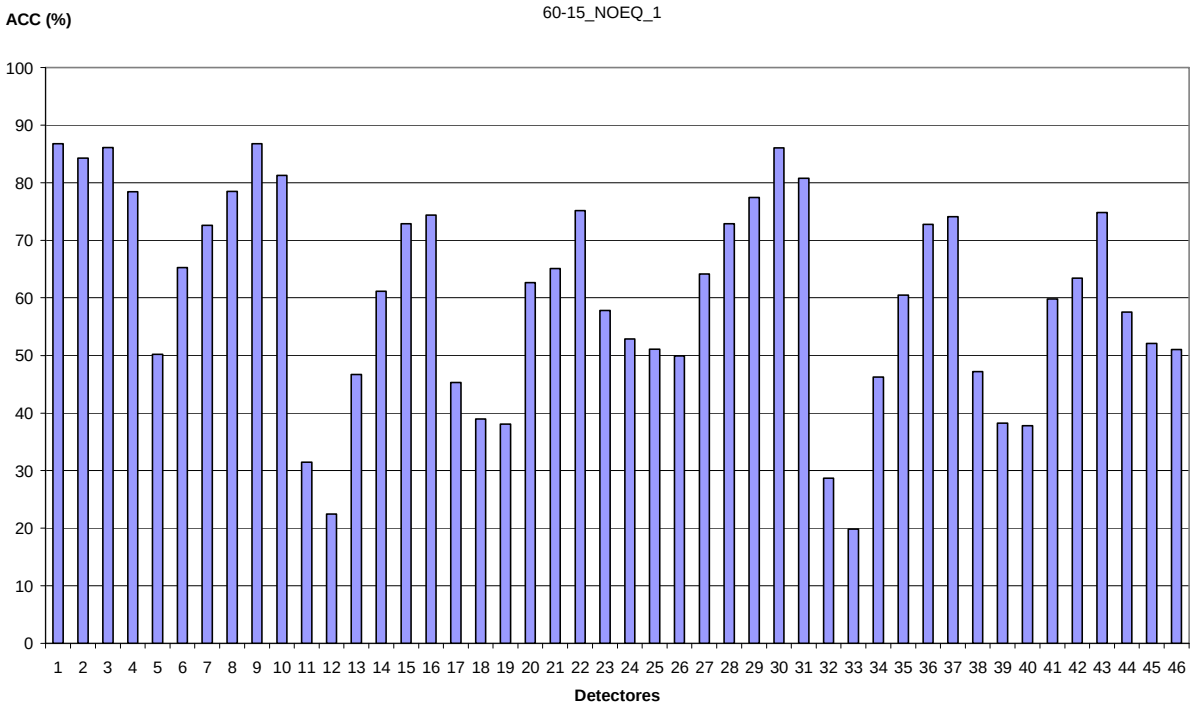


Figura 5.7. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-15, Sin ecualización y escala=1.

Para la situación tamaño de ventana 60-15, sin ecualización y escala 1, siete detectores, el 1, 2, 3, 9, 10, 30 y 31, superan el 80 % pero ninguno de ellos llega la 90%.

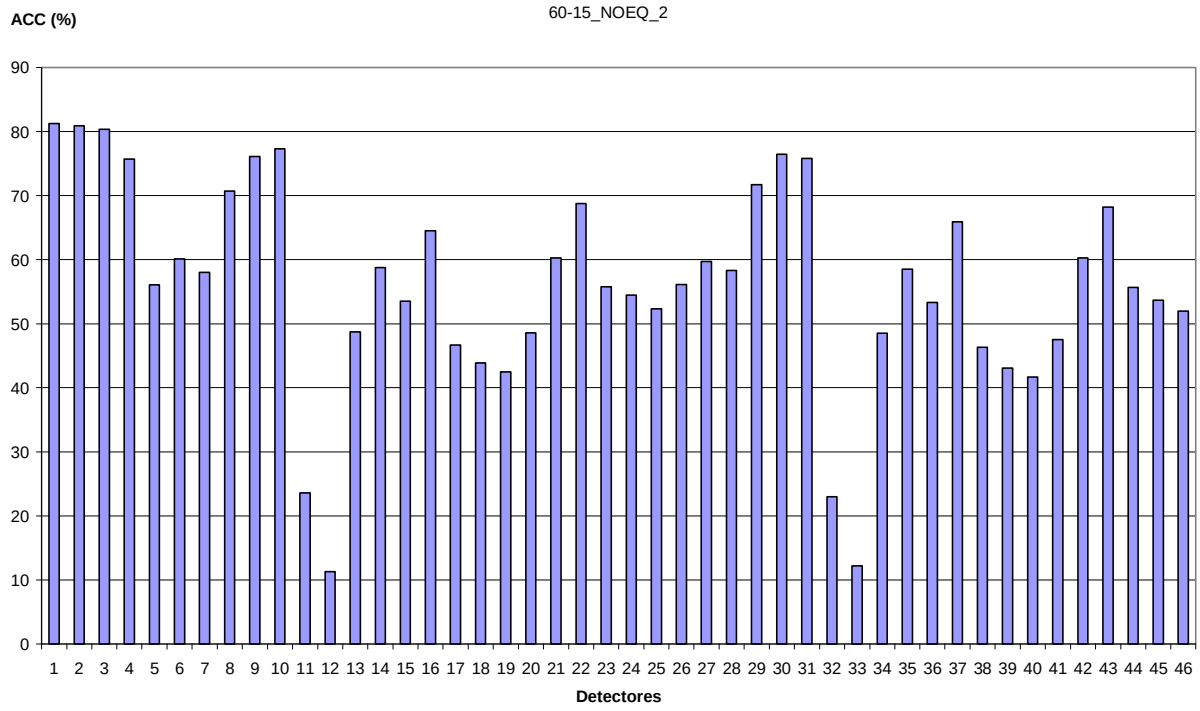


Figura 5.8. Valores de ACC para los 46 detectores con los parámetros: Tamaño de ventana= 60-30, Ecualización y escala=2.

En la figura 5.8, tamaño de venta 60-15, sin ecualización y escala 2, sólo tres detectores, 1, 2 y 3, superan ligeramente el 80%.

Variación de la escala

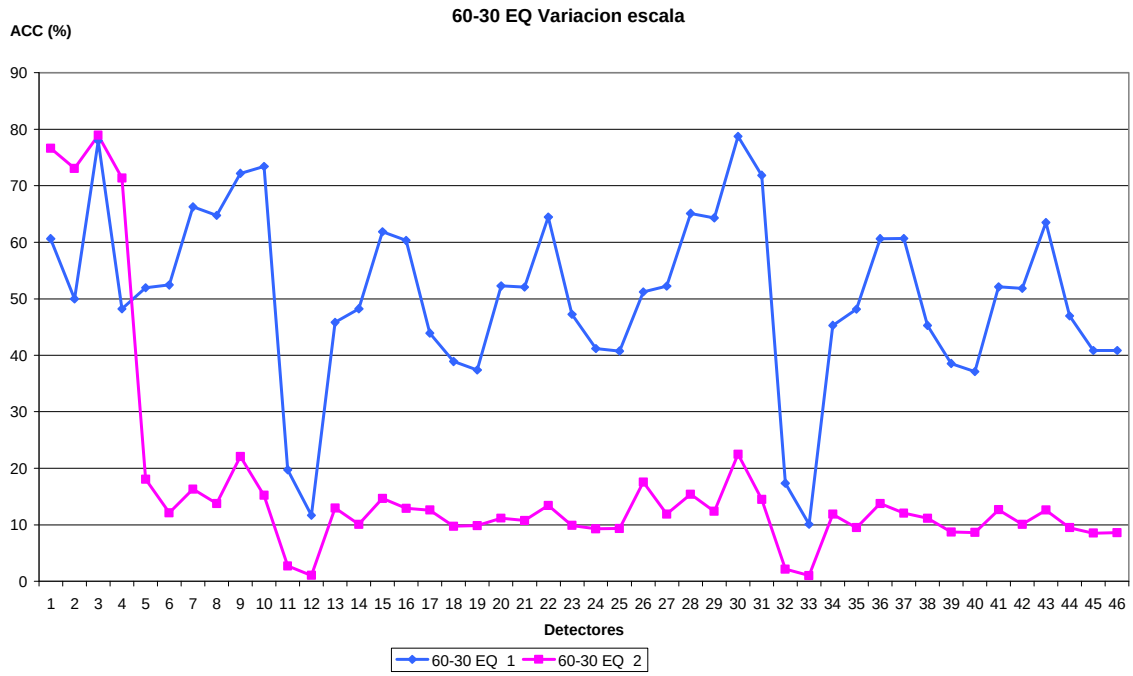


Figura 5.9. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-30 y ecualización y variando la escala; línea azul con escala 1 y línea rosa con escala 2.

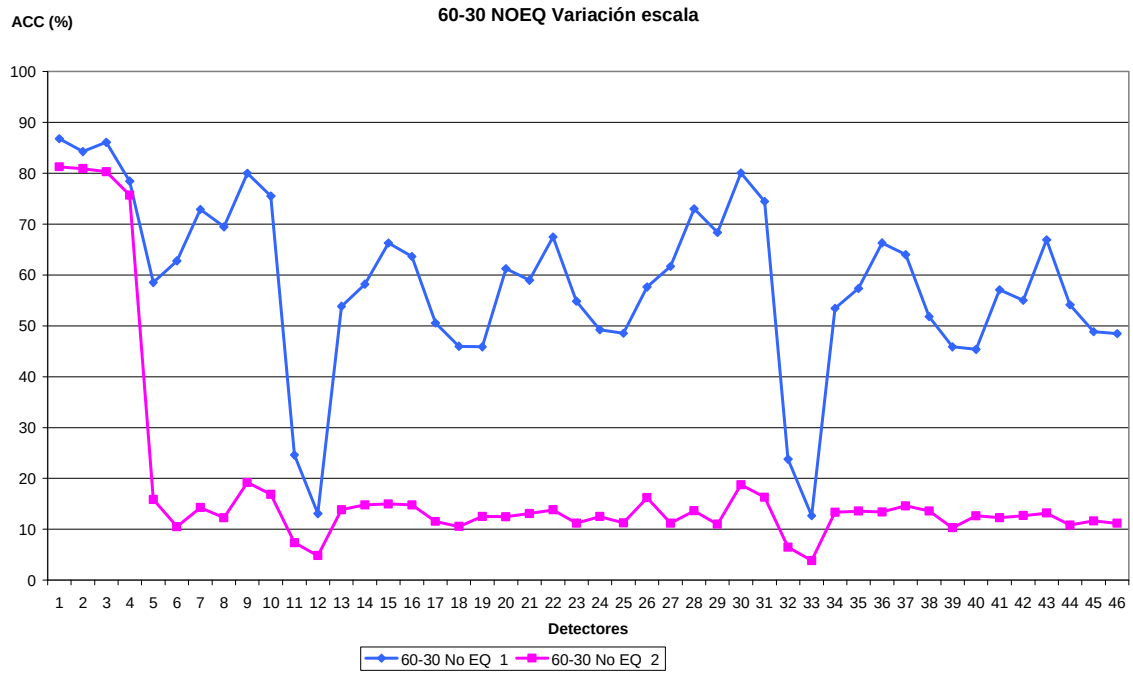


Figura 5.10. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-30 y sin ecualización y variando la escala; línea azul con escala 1 y línea rosa con escala 2.

En la situación tamaño de ventana 60-30, ya sea con o sin ecualización, ninguno de los datos obtenidos para escala 2 supera el 22 % para detectores mixtos.

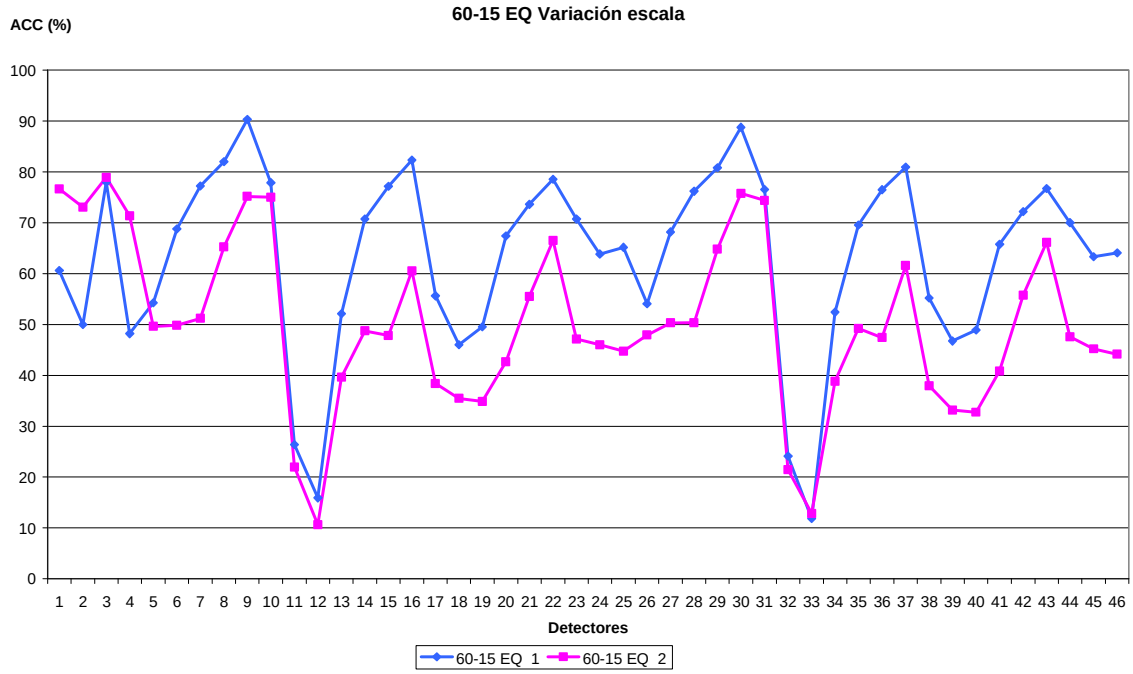


Figura 5.11. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-15 y ecualización y variando la escala; línea azul con escala 1 y línea rosa con escala 2.

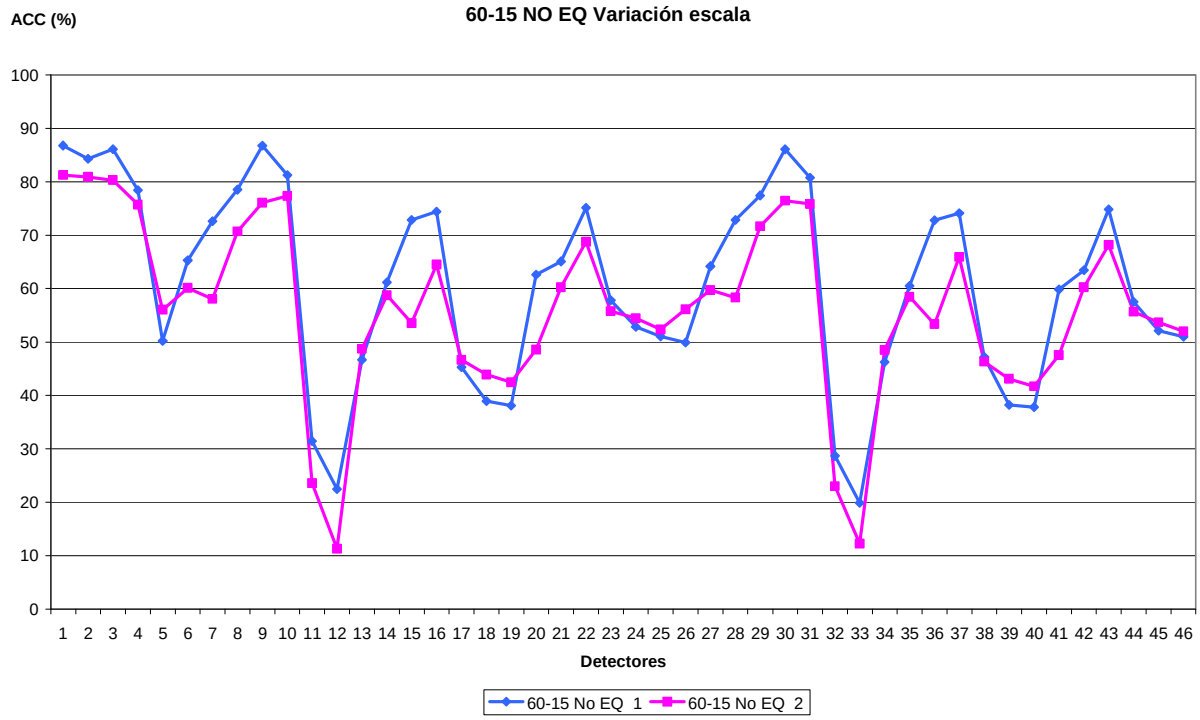


Figura 5.12. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-15 y sin ecualización y variando la escala; línea azul con escala 1 y línea rosa con escala 2.

Para el caso tamaño de ventana 60-15, ya sea con o sin ecualización, la curva de los datos para escala 1 queda un 8% por encima de la curva para escala 2.

Variación ecualización

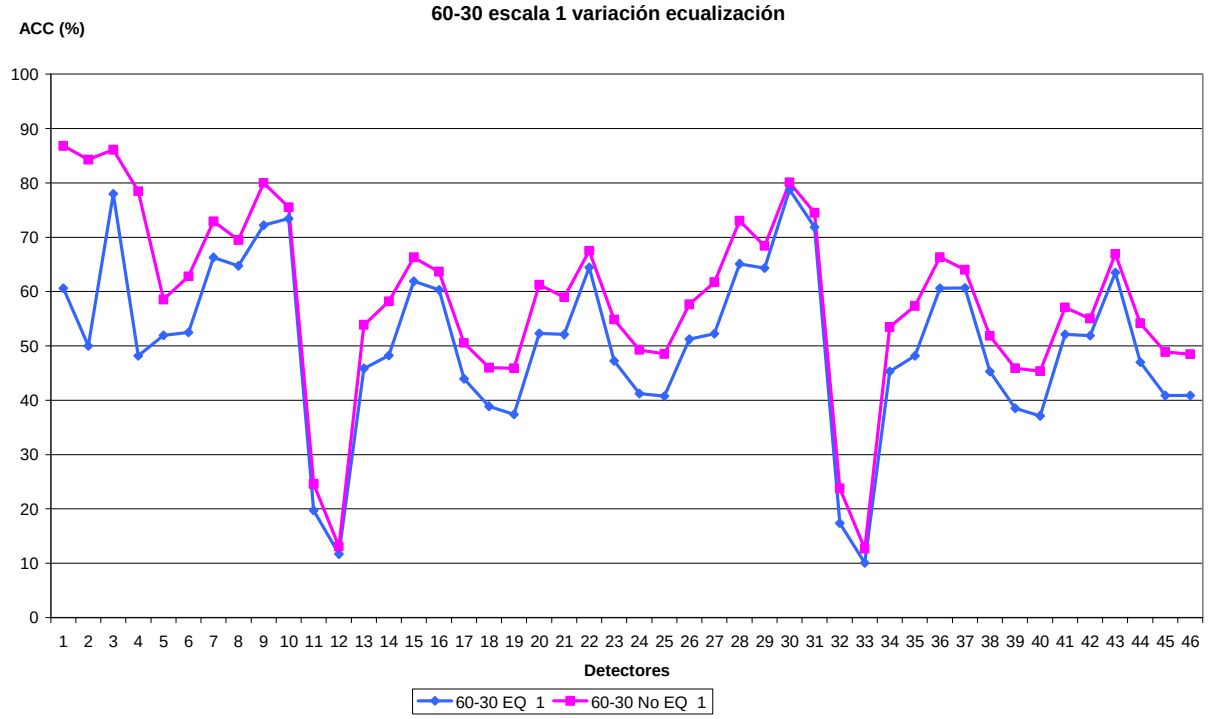


Figura 5.13. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-30 y escala=1 y variando la ecualización; línea azul con ecualización y línea rosa sin ecualización.

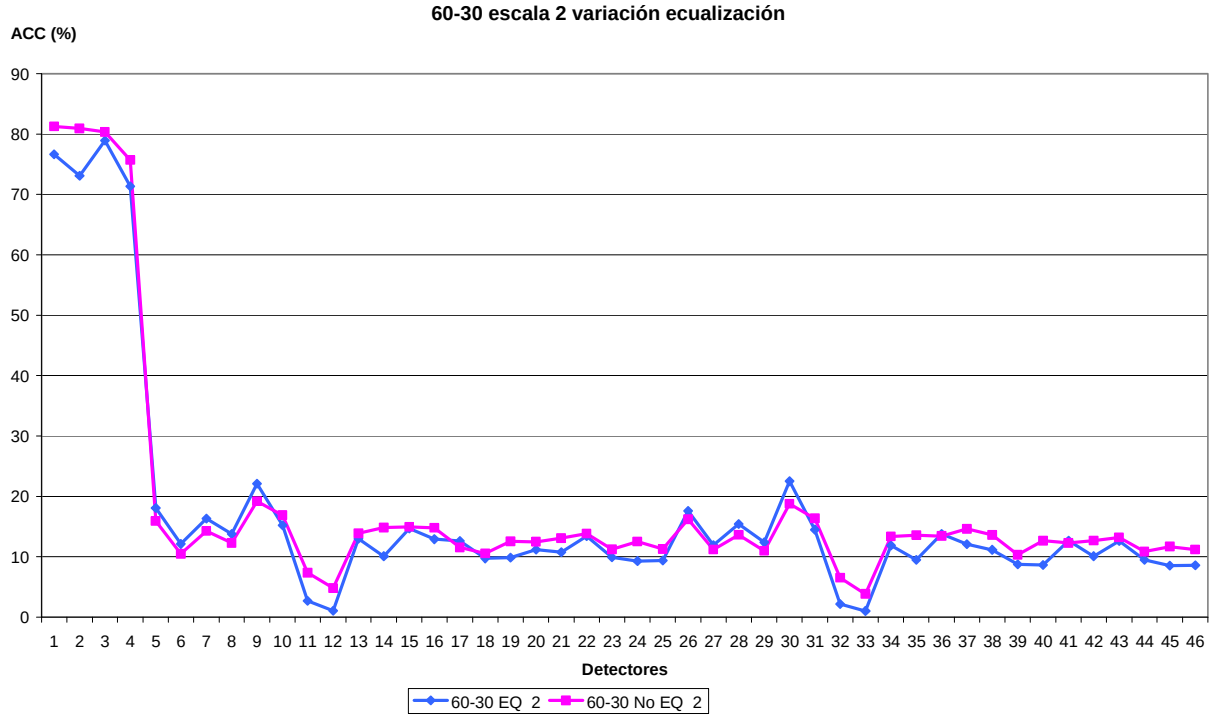


Figura 5.14. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-30 y escala=2 y variando la ecualización; línea azul con ecualización y línea rosa sin ecualización.

Para el caso tamaño de ventana 60-30, ya sea con escala 1 o escala 2, la curva de los datos sin ecualización queda un 5% por encima de la curva con ecualización.

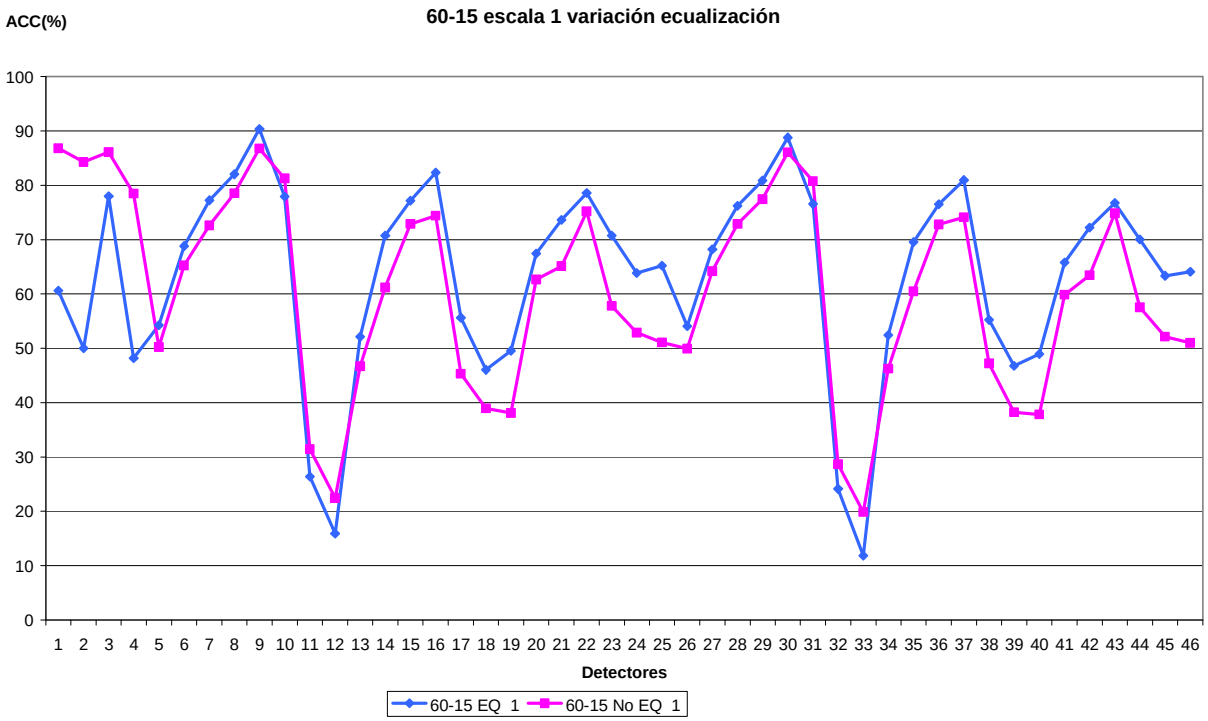


Figura 5.15. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-15 y escala=1 y variando la ecualización; línea azul con ecualización y línea rosa sin ecualización.

En la situación tamaño de ventana 60-15, con escala 1, la curva de los datos con ecualización es ligeramente superior que la curva con ecualización.

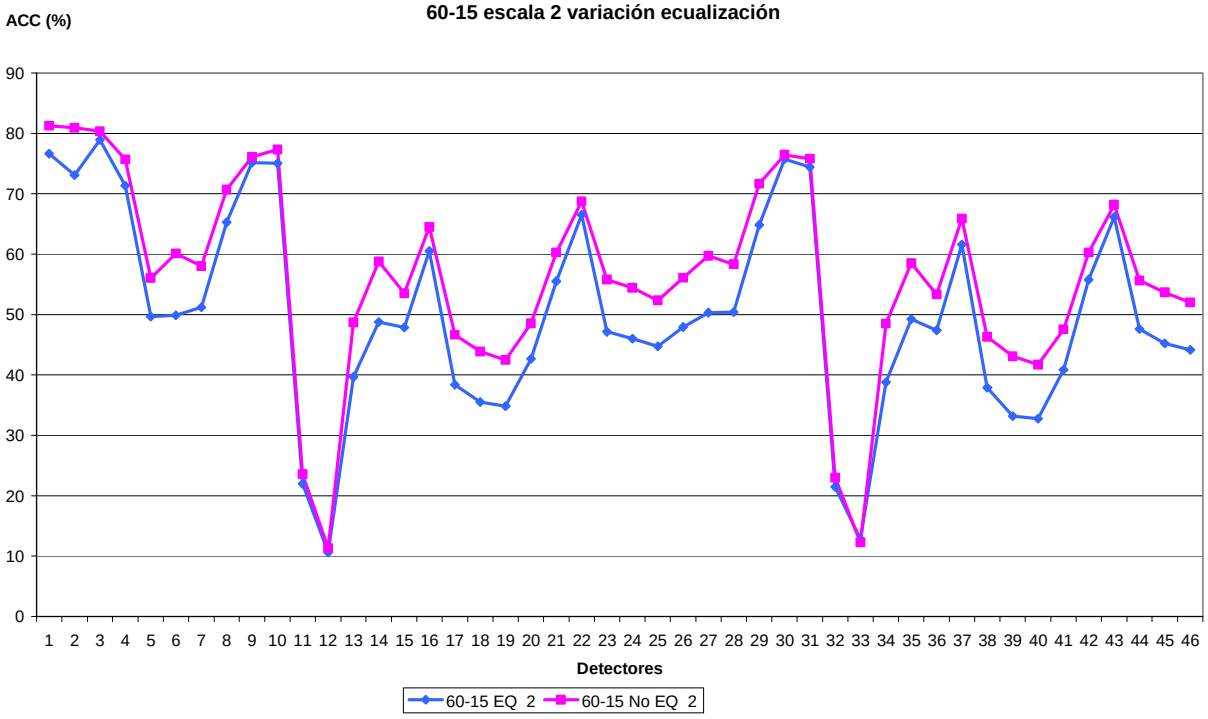


Figura 5.16. Valores de ACC para los 46 detectores fijando los parámetros: Tamaño de ventana= 60-15 y escala=2 y variando la ecualización; línea azul con ecualización y línea rosa sin ecualización.

En la situación tamaño de ventana 60-15, con escala 2, la curva de los datos sin ecualización es ligeramente superior que la curva con ecualización.

Variación del detector de cara para detector mixto

Figura 5.17. Valores de ACC para los detectores mixtos con frontal_face_alt sobre valores de ACC para detectores mixtos con frontal_face_alt2, para todas las combinaciones de parámetros.

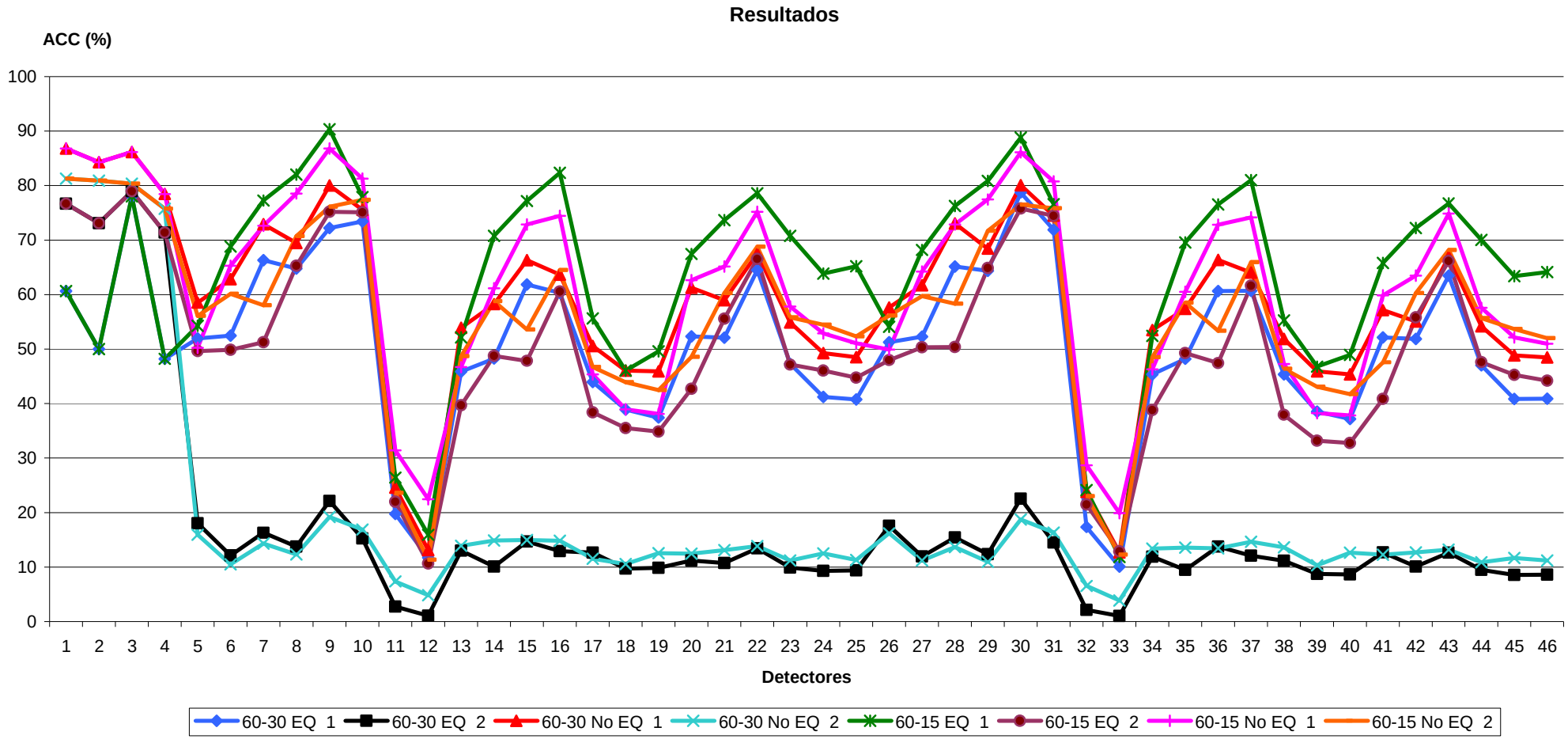


Figura 5.18. Valores de ACC para los 46 detectores para todas las combinaciones de parámetros.

En la figura 5.17 queda patente que para los detectores mixtos, los resultados obtenidos no varían significativamente del uso de haarcascade_frontalface_alt al uso de haarcascade_frontalface_alt2 para la detección de la cara independientemente de la combinación de parámetros empleada.

En la figura 5.18 se observa claramente que los valores de ACC para los detectores 11, 12, 32 y 33 son muy bajos, no superan el 22 %, para ninguna de las combinaciones de parámetros. La combinación de parámetros 60-30, con o sin ecualización y con escala 2 no alcanza el 22% para ninguno de los detectores. Los detectores alcanzan mayores valores de ACC para la combinación de parámetros tamaño de ventana 60-15, con ecualización y escala 1, seguido con un 3% menos de media para todos los detectores de la combinación de parámetros 60-15, sin ecualización y escala 1.

6 CONCLUSIONES Y TRABAJO FUTURO

En este capítulo de la memoria se presentan las conclusiones alcanzadas después del desarrollo de este proyecto. También se plantean algunos trabajos de futuro basados en éste.

6.1 CONCLUSIONES

En la actualidad, la detección de caras es una necesidad para muchas aplicaciones relacionadas con la seguridad, la video conferencia o el ocio. Su importancia crece teniendo en cuenta que es la primera etapa en la mayoría de estas aplicaciones.

Para el desarrollo de este trabajo se ha realizado un estudio del estado del arte (apartado 2) y como queda patente en éste, existen una gran variedad de métodos de detección de caras, muchos de los cuales comparten las mismas técnicas y algoritmos de reconocimiento de objetos. Uno de estos métodos es el implementado por Viola y Jones [1], en el cual está basado este trabajo.

Como se comentó en la introducción del proyecto, el objetivo era el estudio y evaluación efectiva de varios detectores de cara. Para lograr este objetivo se implementó un sistema de evaluación sobre MATLAB a partir del cual se pretendía obtener un análisis sólido del rendimiento de los detectores.

Partiendo del algoritmo de Viola y Jones, las librerías de openCV y la variación de parámetros se han conseguido obtener resultados suficientemente válidos para la detección de caras.

Una vez evaluado el algoritmo se concluyó que con el fin de mejorar la detección lo más efectivo era utilizar detectores de manera consecutiva. El procedimiento consiste en identificar un área como cara para posteriormente evaluar dicha zona utilizando el detector de características faciales. Con esto, si el sistema identifica como característica facial (nariz, boca, ojos…) algún elemento de esa zona, el área seleccionada se cataloga como cara. En caso contrario es desechada. El objetivo principal de encadenar dos detectores consecutivos es disminuir el número de falsos positivos tanto del detector de caras como del detector de características faciales.

El tiempo de trabajo del detector de características faciales se ve al mismo tiempo considerablemente reducido debido a que en vez de analizar la totalidad de la imagen analiza zonas mucho menores. Esto provoca que el sistema propuesto completo no penaliza de manera significativa el rendimiento en términos temporales.

Analizando los resultados obtenidos podemos concluir que el mejor detector para caras es la combinación de detectores haarcascade_frontalface_alt y haarcascade_mcs_nose y los parámetros que optimizan este detector son un tamaño mínimo de ventana de 60 para la cara y de 15 para las características, ecualizando la imagen y sin variar la escala de ésta. De esta manera se alcanzan resultados con una exactitud del 90,34 %.

Los detectores mcs_eyepair_big.xml. y mcs_eyepair_small.xml. presentan resultados muy pobres no alcanzando el 25 % para ninguna combinación de parámetros. De la misma manera, la combinación de parámetros tamaño de ventana 60-30, con o sin ecualización y escala 2 ofrece resultados no convincentes, sin alcanzar tampoco el 25 % para los 46 detectores evaluados.

En líneas generales se puede concluir que si no variamos la escala de las imágenes se obtienen mejores resultados. Igualmente para tamaño 60-15 es mejor ecualizar las imágenes mientras que para tamaño 60-30 es mejor no hacerlo.

El cambio de detector de cara de haarcascade_frontalface_alt.xml a haarcascade_frontalface_alt2.xml no varía significativamente los resultados para ACC.

Es reseñable el hecho de que si la evaluación de los detectores se hubiera realizado sobre otra base de datos de prueba más acorde con la finalidad de eye-tracking, es decir, con imágenes de cara más cercanas y frontales, los resultados obtenidos serian incluso mejores.

6.2 TRABAJO FUTURO

En este proyecto únicamente se ha trabajado con detección facial frontal por lo que posibles líneas futuras podrían estar enfocadas en detección de caras y características faciales cuyas imágenes son de perfil.

Para posibles trabajos futuros sería muy interesante realizar un entrenamiento del detector con mejor rendimiento sobre la base de datos realizada para mejorar la efectividad del mismo. La librería OpenCV contiene programas o comandos que pueden ser utilizados para entrenar clasificadores, por lo tanto es posible crear clasificadores propios haciendo uso de éstas. (Su uso queda explicado en anexo 5).

Una vez evaluado el rendimiento de la técnica descrita anteriormente, la culminación del sistema consistiría en introducir un sistema de reconocimiento facial.

7 BIBLIOGRAFRÍA

1. PAUL VIOLA and MICHAEL J. JONES, “Robust Real-Time Face Detection“, Julio 2003.

2. Danijela Vukadinovic and Maja Pantic, “Fully Automatic Facial Feature Point Detection Using Gabor Feature Based Boosted Classifiers”, Hawaii October 2005.

3. Padhraic Smyth, “Face Detection using the Viola-Jones Method”, Department of Computer Science, University of California, Irvine, 2007.

4. Theo Ephraim Tristan Himmelman Kaleem Siddiqi, “Real-Time Viola-Jones Face Detection in aWeb Browser”, School of Computer Science & Centre for Intelligent Machines, McGill University, 2009.

5. Fermí Vilà, Programación en C/C++ (Manual FV).

6. Víctor Domínguez Báguena y Mª Luisa Rapún Banzo. “Matlab en cinco lecciones de Numérico”. Febrero de 2006.

7. Javier García de Jalón, José Ignacio Rodríguez, Jesús Vidal. “Aprenda Matlab 7.0 como si estuviera en primero”. Madrid, diciembre 2005.

8. Antonio Rama, Francesc Tarrés, “Un nuevo método para la detección de caras basado en Integrales Difusas”, Dept. Teoría del Senyal i Comunicacions - Universitat Politècnica de Catalunya, Barcelona.

9. G. Yang, T.S. Huang, “Human Face Detection in a Complex Background Pattern Recognition”, 1994.

10. Ming-Hsuan Yang, Member, IEEE, David J. Kriegman, Senior Member, IEEE, and Narendra Ahuja, Fellow, IEEE, “Detecting Faces in Images: A Survey”, Enero 2002.

11. C. Kotropoulos and I. Pitas, “Rule-Based Face Detection in Frontal Views”, Proc. Int’l Conf. Acoustics, Speech and Signal Processing, 1997.

12. C. Kotropoulos and I. Pitas,”Using Support Vector Machines to Enhance the Performance of Elastic Graph Matching for Frontal Face Authentication” , IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, July 2001.

13. Henry A. Rowley Shumeet Baluja Takeo Kanade, “Human Face Detection in Visual Scenes”, School of Computer Science Carnegie Mellon University, Pittsburgh, November 1995.

14. Ming-Hsuan Yang and Narendra Ahuja “Detecting Human Faces in Color Images”, Institute and Department of Electrical and Computer Engin, University of Illinois.

15. S.A. Sirohey, “Human Face Segmentation and Identification”, Technical Report, Univ. of Maryland, 1993.

16. D. Chetverikov and A. Lerch, “Multiresolution Face Detection”, Theoretical Foundations of Computer Vision, 1993.

17. M.C. Burl, T.K. Leung, and P. Perona, “Face Localization via Shape statistics”, Proc. First Int’l Workshop Automatic Face and Gesture Recognition, pp. 154-159, 1995

18. K.C. Yow and R. Cipolla, “A Probabilistic Framework for Perceptual Grouping of Features for Human Face Detection”, Proc. Second Int’l Conf. Automatic Face and Gesture Recognition, pp. 16-21, 1996.

19. Y. Amit, D. Geman, and B. Jedynak, “Efficient Focusing and Face Detection”, Face Recognition: From Theory to Applications, H. Wechsler, P.J. Phillips, V. Bruce, F. Fogelman-Soulie, and T.S. Huang, eds., vol. 163, pp. 124-156, 1998.

20. M.F. Augusteijn and T.L. Skujca, “Identification of Human Faces through Texture-Based Feature Recognition and Neural Network Technology”, Proc. IEEE Conf. Neural Networks, pp. 392-398, 1993.

21. R.M. Haralick, K. Shanmugam, and I. Dinstein, “Texture Features for Image Classification”, IEEE Trans. Systems, Man, and Cybernetics, vol. 3, no. 6, pp. 610-621, 1973.

22. Y. Dai and Y. Nakano, “Face-Texture Model Based on SGLD and Its Application in Face Detection in a Color Scene,” Pattern Recognition, vol. 29, no. 6, pp. 1007-1017, 1996.

23. D. Saxe and R. Foulds, “Toward Robust Skin Identification in Video Images”, Proc. Second Int’l Conf. Automatic Face and Gesture Recognition, pp. 379-384, 1996.

24. R. Kjeldsen and J. Kender, “Finding Skin in Color Images”, Proc. Second Int’l Conf. Automatic Face and Gesture Recognition, pp. 312- 317, 1996.

25. S. McKenna, Y. Raja, and S. Gong, “Tracking Colour Objects Using Adaptive Mixture Models”, Image and Vision Computing, vol. 17, nos. 3/4, pp. 223-229, 1998.

26. Q. Chen, H. Wu, and M. Yachida, “Face Detection by Fuzzy Matching”, Proc. Fifth IEEE Int’l Conf. Computer Vision, pp. 591-596, 1995.

27. K. Sobottka and I. Pitas, “Face Localization and Feature Extraction Based on Shape and Color Information”, Proc. IEEE Int’l Conf. Image Processing, pp. 483-486, 1996.

28. E. Saber and A.M. Tekalp, “Frontal-View Face Detection and Facial Feature Extraction Using Color, Shape and Symmetry Based Cost Functions”, Pattern Recognition Letters, vol. 17, no. 8, pp. 669- 680, 1998.

29. S.-H. Kim, N.-K. Kim, S.C. Ahn, and H.-G. Kim, “Object Oriented Face Detection Using Range and Color Information”, Proc. Third Int’l Conf. Automatic Face and Gesture Recognition, pp. 76-81, 1998.

30. E. Saber, A.Murat Tekalp, “Frontal-View Face Detection and Facial feature Extraction using Color, Shape and Symmetry Based Cost Functions”, 1998.

31. P. Sinha, “Object Recognition via Image Invariants: A case Study”, Investigative Ophthalmology and Visual Science, Vol. 35, pp. 1735-1740. 1994.

32. A. Yuille, P. Hallinan and D. Cohen, “Feature Extraction from Faces Using Deformable Templates”, Int’l J. Computer Vision, vol. 8, no. 2, pp. 99-111, 1992.

33. H. Rowley, S. Baluja and T. Kanade, “Neural Network-Based Face Detection”, Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 203-208, 1996.

34. M.-H. Yang, D. Roth, and N. Ahuja, “A SNoW-Based Face Detector”, Advances in Neural Information Processing Systems 12, S.A. Solla, T. K. Leen, and K.-R. Muller, eds., pp. 855-861, MIT Press, 2000.

35. T. V. Pham, M. Worring, A. W. M. Smeulders, “Face detection by aggregated Bayesian network classifiers”, Pattern Recognition Letters, Vol. 23, pp. 451-461, 2001.

36. H. Schneiderman and T. Kanade, “A Statistical Method for 3D Object Detection Applied to Faces and Cars”, Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 746-751, 2000.

37. Y. Freund, R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to Boosting”, Computational Learning Theory: Eurocolt 95, ed. Springer- Verlag, pp. 23-37, 1995.

38. G. Bradski, A. Kaehler, “Learning OpenCV: Computer Vision with the OpenCV Library”, Cambridge : O'REILLY, 2008, ISBN 978-0-596-51613-0.

39. http://www.bioid.com/support/downloads/software/bioid-face-database.html

40. http://www.ew2008.org/matlab/techdoc/matlab_external/ch04crea.html

41. http://opencv.willowgarage.com/wiki/FaceDetection

42. http://facefinderarm.wordpress.com/

43. http://www.face-rec.org/interesting-papers/

44. http://www.xavigimenez.net/blog/2010/02/face-detection-how-to-find-faces-with-opencv/

45.http://help.adobe.com/es_ES/InCopy/5.0/help.html?content=WSa285fff53dea4f8617383751001ea8cb3f-7404.html

46. http://www.sfr-fresh.com/

47.http://www710.univlyon1.fr/~bouakaz/OpenCV0.9.5/docs/ref/OpenCVRef_Experimental.htm

48. http://www.mathworks.com/matlabcentral/fileexchange/19912-open-cv- viola-jones-face-detection-in-matlab

49. http://www.nosolousabilidad.com/articulos/eye-tracking.htm

50. http://ieeexplore.ieee.org/Xplore/dynhome.jsp

51. http://www.sfr-fresh.com/unix/misc/OpenCV-2.1.0.tar.gz:a/OpenCV-2.1.0/

52. http://www.face-rec.org/databases/

53. file:////tscserver/videosoffaces$/

54. http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html

55. http://www.anefian.com/research/face_reco.htm

56. http://www.vision.caltech.edu/html-files/archive.html

8 ANEXOS

8.1 ANEXO 1. FUNCIONES EN MATLAB

8.1.1 etiquetar.m

```
%ETIQUEDADO NUEVA BASE DE DATOS

carpetaspts=dir('*pts*');
carpetasf=dir('*f*');
NumCarpetas=size(carpetasf,1);
% NumCarpetaspts=size(carpetaspts,1);

for t=1:NumCarpetas
    cd(carpetasf(t).name);
    Images=dir('*B*.pgm');
    cd ..;

    NumImages=size(Images,1);
    Mat=[];
    MatrizPositions=[];

    cd(carpetaspts(t).name);
    positions=dir('./*.pts');
    cd ..;

for i=1:NumImages
    cd(carpetaspts(t).name);
    B=positions(i).name;
    fid=fopen(B);
for w=1:24
    tline=fgets(fid);
    aux=str2num(tline);
    Mat=[Mat;aux];
end
fclose(fid);

cd ..;
cd(carpetasf(t).name);
img=imread(Images(i).name);

a0=Mat(10,1)-15;
b0=Mat(10,2)-25;
c0=Mat(13,1)+15;
d0=Mat(20,2)-20;

a1=Mat(10,1)-22;
b1=Mat(6,2)-12;
c1=Mat(11,1)+15;
d1=Mat(11,2)+10;

a2=Mat(12,1)-15;
b2=Mat(7,2)-12;
c2=Mat(13,1)+22;
d2=Mat(13,2)+12;
```

```
a3=Mat(16,1)-12;
b3=Mat(11,2);
c3=Mat(17,1)+12;
d3=Mat(17,2)+10;

a4=Mat(3,1)-8;
b4=Mat(18,2)-5;
c4=Mat(4,1)+10;
d4=Mat(19,2)+5;

Mat=[];

Rect0=[a0 b0;c0 b0;c0 d0;a0 d0;a0 b0];
Rect1=[a1 b1;c1 b1;c1 d1;a1 d1;a1 b1];
Rect2=[a2 b2;c2 b2;c2 d2;a2 d2;a2 b2];
Rect3=[a3 b3;c3 b3;c3 d3;a3 d3;a3 b3];
Rect4=[a4 b4;c4 b4;c4 d4;a4 d4;a4 b4];

figure(1);
imshow(img);
truesize;
hold on;

plot (Rect0(:,1), Rect0(:,2), 'g');
plot (Rect1(:,1), Rect1(:,2), 'g');
plot (Rect2(:,1), Rect2(:,2), 'g');
plot (Rect3(:,1), Rect3(:,2), 'g');
plot (Rect4(:,1), Rect4(:,2), 'g');

Matriz=[a0 b0 c0 d0 a1 b1 c1 d1 a2 b2 c2 d2 a3 b3 c3 d3 a4 b4 c4 d4];
MatrizPositions=[MatrizPositions;Matriz];
cd ..;
% pause
end

save(strrep('C:\Documents and
Settings\rsola\Escritorio\LBL\LabelsFaceNAME.txt','NAME',carpetasf(t).
name),'MatrizPositions','-ascii','-tabs');

end
```

8.1.2 CambiarNombre

```
cont=1;
for cent=0:1
    for dec=0:9
        for un=0:9
            ma(cont,:)=[0 cent dec un];
            cont=cont+1;
            if cont==152
                return
            end
        end
    end
end

Suf='.jpg';
carpetas=dir('*f*');
```

```
NumCarpetas=size(carpetas,1);

for t=1:NumCarpetas
    aux='Name_';
    Pref=strrep(aux, 'Name',carpetas(t).name);
    cd(carpetas(t).name);
    Images=dir('*B*.pgm');
    NumImages=size(Images,1);
    for j=1:NumImages
        str=num2str(ma(j,:));
        B=strcat(Pref,str,Suf);
        copyfile(Images(j).name,B);
    end
    movefile('f*',strrep('C:\Documents and
Settings\rsola\Escritorio\images\NAME','NAME',carpetas(t).name),'f');
    cd ..;
end
```

8.1.3 Labels

Entradas: i y NumImages
Salidas: MatrizPositions

- i
  Indica el numero de archivo txt a leer.

- NumImages
  Numero de imágenes que contiene la carpeta a la que corresponde el archivo txt.

- MatrizPositions
  Matriz que contiene las posiciones de los diferentes elementos (cara y/o ojos y/o nariz y/o boca).

```
%labels

function MatrizPositions=labels(i,NumImages)
 %lee los archivos txt de las posiciones y coloca cada archivo en una
 %matriz
 %Inputs
 %i=indica el numero de archivo txt a leer
 %NumImages=Numero de imagenes que contiene la carpeta a la que
corresponde
 %el archivo txt
 %Outputs
 %MatrizPositions=Matriz que contiene las posiciones de los diferentes
 %elementos

Labels=dir('*LabelsFace*');
cd(Labels.name);
positions=dir('./*.txt');
S=[];

B=positions(i).name;
fid = fopen(B, 'r');
a= fscanf(fid, '%e', [20 NumImages]);
fclose(fid);
```

```
S=[S;a];
MatrizPositions=S';
cd ..;
end
```

8.1.4 FaceDetection

Entradas: img, Array y FacePositions.
Salidas: Rectangle, Rectanglemix, RectangleFeature y r.

- Img
  Imagen de entrada en la que se detectará la cara y/o nariz y/o ojo izquierdo y/o ojo derecho y/o boca.

- Array

Definido en el script *Test.m*, quedan recogidos los parámetros para las diferentes detecciones. Es donde se indica el o los xml a usar.

- FacePositions
  Posicion real donde se encuentra la cara en la imagen, está etiquetada en un archivo txt, *LabelsFace.txt* (parte de *MatrizPositions*).

- Rectangle
  Matriz que contiene las posiciones de los diferentes rectángulos detectados, posibles candidatos, cuando el detector no es mixto.

- r
  Indica el número de características que detecta. (si r>1 detector mixto).

- Rectanglemix
  Matriz que contiene las posiciones de los diferentes rectángulos detectados como cara, rectángulos candidatos a cara.

- RectangleFeature
  Matriz que contiene las posiciones de los diferentes rectángulos detectados como características faciales, rectángulos candidatos a característica facial.

```
%FaceDetection

% Array=Datos;
% MatrizPositions=labels(i,NumImages)
% FacePositions=MatrizPositions(i,1:4);


function [Rectangle Rectanglemix RectangleFeature
r]=FaceDetection(img,Array,FacePositions)
%inputs:
%img: imagen de entrada en la que se detectará la cara y/o nariz y/o
ojo
%izquierdo y/o ojo derecho y/o boca
%Array:definido en el script, quedan recogidos los parametros para las
%diferentes detecciones
```

```
%FacePositions: Posicion real donde se encuentra la cara en la imagen,
está
%etiquetada en un archivo txt

%outputs:
%Rectangle: Matriz que contiente las posiciones de los diferentes
%rectangulos detectados cuando el detector no es mixto
%r: Indica el número de caracteristicas que detecta. (si r>1 detecctor
mixto)
%Rectanglemix: Matriz que contiene las posiciones de los diferentes
%rectangulos detectados como cara.
%RectangleFeature: Matriz que contiente las posiciones de los
diferentes
%rectangulos detectados como caracteristicas faciales.

r=0;
A=size(Array,2);
RectangleFeature=cell(1,A);
Rectangleaux=[];
Rectangle1=[];
Rectangle2=[];
Rectanglemov=[];
dim=size(img,3);

if dim==3
    Img=rgb2gray(img);
%     Img=uint8(Img);
elseif dim==1
    Img=img;
%     Img=double(img);
%     Img=uint8(Img);
end

for s=1:A
    if size(Array{1,s},1)==1
        r=r+1;
    end
end

if r==1;

figure(1);
imshow(img);
truesize;
hold on;

    for i=1:A
        if i==1
            if size(Array{1,i},1)==1
               Face = cvod200uint_v02(Array{i}.xml,Img,1,0,60);
               SizeFace=size(Face,1);

               if Face~=-1
                    for j=1:SizeFace
                        Rectangle1aux = [Face(j,1) Face(j,2);
Face(j,1)+Face(j,3) Face(j,2); Face(j,1)+Face(j,3)...
                            Face(j,2)+Face(j,4); Face(j,1)
Face(j,2)+Face(j,4); Face(j,1) Face(j,2)];
                        Rectangle1=[Rectangle1,Rectangle1aux];
                    end
                    for g=1:2:size(Rectangle1,2)
```

```
                        plot(Rectangle1(:,g), Rectangle1(:,g+1), 'g');
                    end
                    Rectangle=Rectangle1;
                else
                  Rectangle=NaN(5,2);
                  'Cara no detectada'
                end
            end
        else
           if size(Array{1,i},1)==1
               x=FacePositions(1,1)-30;
               y=FacePositions(1,2)-30;
               w=FacePositions(1,3)+30-FacePositions(1,1)+30;
               h=FacePositions(1,4)+30-FacePositions(1,2)+30;
               croppedimg=imcrop(img,[x y w h]);

               if dim==3
                   croppedimggray=rgb2gray(croppedimg);
               elseif dim==1
                   croppedimggray=croppedimg;
               end

%                Face=FaceDetect(Array{i}.xml,croppedimggray);
               Face =
cvod200uint_v02(Array{i}.xml,croppedimggray,1,0,15);
               SizeFace=size(Face,1);
               if Face~=-1
                   for j=1:SizeFace
                        Rectangle2aux = [Face(j,1) Face(j,2);
Face(j,1)+Face(j,3) Face(j,2); ...
                            Face(j,1)+Face(j,3) Face(j,2)+Face(j,4);
Face(j,1)  Face(j,2)+Face(j,4); ...
                            Face(j,1) Face(j,2)];
                        Rectangle2=[Rectangle2,Rectangle2aux];
                   end
                   for j=1:2:size(Rectangle2,2)

plot(Rectangle2(:,j)+x,Rectangle2(:,j+1)+y,'g');
                       Rectanglemovaux=[Rectangle2(:,j)+x,
Rectangle2(:,j+1)+y];
                       Rectanglemov=[Rectanglemov,Rectanglemovaux];
                   end
                    Rectangle=Rectanglemov;
               else
                    Rectangle=NaN(5,2);
               end
           end
        end
    end

hold off;

Rectanglemix=NaN(5,r*2);
RectangleFeature{1,A}=NaN(5,2);


%si detector mixto
else
    figure(1);
    imshow(img);
    truesize;
```

```
    hold on;

%     Face=FaceDetect('haarcascade_frontalface_alt.xml',Img);
%     Face =
cvod200uint_v02('haarcascade_frontalface_alt.xml',Img,2,0);
    Face = cvod200uint_v02(Array{1}.xml,Img,1,0,60);
    SizeFace=size(Face,1);
    if Face~=-1
        for j=1:SizeFace
            Rectangle1aux = [Face(j,1) Face(j,2); Face(j,1)+Face(j,3)
Face(j,2); Face(j,1)+Face(j,3)...
                Face(j,2)+Face(j,4); Face(j,1)  Face(j,2)+Face(j,4);
Face(j,1) Face(j,2)];
            Rectangle1=[Rectangle1,Rectangle1aux];
        end
        for g=1:2:size(Rectangle1,2)
            plot(Rectangle1(:,g), Rectangle1(:,g+1), 'g');
        end

%        bucle: tendré k recortar todos los rectangulos k me devuelva
el
%        face--> tendré varias croppedimg, tantas como rectangulos
       croppedimg=cell(1,SizeFace);
        for f=1:SizeFace
            x=Face(f,1);
            y=Face(f,2);
            w=Face(f,3);
            h=Face(f,4);
            croppedimg{1,f}=imcrop(img,[x y w h]);

            if dim==3
                croppedimggray{1,f}=rgb2gray(croppedimg{1,f});
            elseif dim==1
                croppedimggray{1,f}=croppedimg{1,f};
            end

            for i=2:A
                if size(Array{1,i},1)==1
    %               Face=FaceDetect(Array{i}.xml,croppedimggray);
                   Feature =
cvod200uint_v02(Array{i}.xml,croppedimggray{1,f},1,0,15);
                   SizeFeature=size(Feature,1);
                   if Feature~=-1
                       Rectangle2=[];
                       for j=1:SizeFeature
                           Rectangle2aux = [Feature(j,1) Feature(j,2);
Feature(j,1)+Feature(j,3) Feature(j,2);...
                               Feature(j,1)+Feature(j,3)
Feature(j,2)+Feature(j,4); Feature(j,1)  ...
                               Feature(j,2)+Feature(j,4); Feature(j,1)
Feature(j,2)];
                           Rectangle2=[Rectangle2,Rectangle2aux];
                       end
                       Rectanglemov=[];
                       for j=1:2:size(Rectangle2,2)

plot(Rectangle2(:,j)+x,Rectangle2(:,j+1)+y,'g');
                            Rectanglemovaux=[Rectangle2(:,j)+x,
Rectangle2(:,j+1)+y];

Rectanglemov=[Rectanglemov,Rectanglemovaux];
```

```
                        end
                        RectangleFeatureaux=Rectanglemov;
                    else
                        RectangleFeatureaux=NaN(5,2);
                    end

RectangleFeature{1,i}=[RectangleFeature{1,i},RectangleFeatureaux];
                end
            end
        end
        Rectanglemix=[Rectangle1,Rectangleaux];
        Rectangle=NaN(5,r*2);

        for s=1:A
            if size(RectangleFeature{1,s},1)~=0
                if ~isnan(RectangleFeature{1,s}(:,:))==1
                elseif isnan(RectangleFeature{1,s}(:,:))==1
                    RectangleFeature{1,s}=[];
                else
                        AUX=(isnan(RectangleFeature{1,s}(:,:)));
                        d=find(AUX);
                        column1=(d(1,1)-1)/5;
                        column2=d(size(d,1))/5;

RectangleFeature{1,s}=[RectangleFeature{1,s}(:,1:column1) ...

RectangleFeature{1,s}(:,column2+1:size(RectangleFeature{1,s},2))];
                end
            end
        end

    else
        'Cara no detectada'
        Rectangle=NaN(5,r*2);
        RectangleFeature{1,1}=NaN(5,2);
        Rectanglemix=NaN(5,r*2);
    end
    hold off;
end
end
```

8.1.5 Test

```
%Test  (script)

%Datos{i}.percents=[Pit Pib Pil Pir; Pot Pob Pol Por];

%Pit= porcentaje interior superior
%Pib= porcentaje interior inferior
%Pil= porcentaje interior izquierdo
%Pir= porcentaje interior derecho
%Pot= porcentaje exterior superior
%Pob= porcentaje exterior inferior
%Pol= porcentaje exterior izquierdo
%Por= porcentaje exterior derecho

%it= margen interior superior
%ib= margen interior inferior
%il= margen interior izquierdo
%ir= margen interior derecho
```

```
%ot= margen exterior superior
%ob= margen exterior inferior
%ol= margen exterior izquierdo
%or= margen exterior derecho

%Los porcentajes, y por tanto los margenes, son relativos al tamaño de
la
%cara

Datos{1}.xml='haarcascade_frontalface_alt.xml';
Datos{1}.percents=[10/100 10/100 5/100 5/100; 75/100 50/100 50/100
50/100];
Datos{1}.parent='Nan';
%
% Datos{1}.xml='haarcascade_frontalface_alt2.xml';
% Datos{1}.percents=[10/100 10/100 5/100 5/100; 75/100 50/100 50/100
50/100];
% Datos{1}.parent='Nan';
%
% Datos{1}.xml='haarcascade_frontalface_alt_tree.xml';
% Datos{1}.percents=[10/100 10/100 5/100 5/100; 75/100 50/100 50/100
50/100];
% Datos{1}.parent='Nan';
%
% Datos{1}.xml='haarcascade_frontalface_default.xml';
% Datos{1}.percents=[10/100 10/100 5/100 5/100; 75/100 50/100 50/100
50/100];
% Datos{1}.parent='Nan';

% Datos{2}.xml='haarcascade_righteye_2splits.xml';
% Datos{2}.percents=[55/100 20/100 40/100 45/100; 75/100 50/100 50/100
50/100];
% Datos{2}.parent=Datos{1};
% %
% % Datos{2}.xml='haarcascade_mcs_righteye.xml';
% % Datos{2}.percents=[55/100 20/100 40/100 45/100; 75/100 50/100
50/100 50/100];
% % Datos{2}.parent=Datos{1};
%
% Datos{3}.xml='haarcascade_lefteye_2splits.xml';
% Datos{3}.percents=[55/100 20/100 45/100 40/100; 75/100 50/100 50/100
50/100];
% Datos{3}.parent=Datos{1};

% Datos{3}.xml='haarcascade_mcs_lefteye.xml';
% Datos{3}.percents=[55/100 20/100 45/100 40/100; 75/100 50/100 50/100
50/100];
% Datos{3}.parent=Datos{1};
%
Datos{4}.xml='haarcascade_mcs_nose.xml';
Datos{4}.percents=[75/100 15/100 20/100 20/100; 30/100 75/100 48/100
48/100];
Datos{4}.parent=Datos{1};
%
% Datos{5}.xml='haarcascade_mcs_mouth.xml';
% Datos{5}.percents=[20/100 20/100 45/100 45/100; 40/100 40/100 90/100
90/100];
% Datos{5}.parent=Datos{1};

% Datos{6}.xml='haarcascade_mcs_eyepair_big.xml';
```

```
% Datos{6}.percents=[20/100 5/100 40/100 40/100; 30/100 30/100 30/100
30/100];
% Datos{6}.parent=Datos{1};
% %
% Datos{6}.xml='haarcascade_mcs_eyepair_small.xml';
% Datos{6}.percents=[20/100 5/100 40/100 40/100; 30/100 30/100 30/100
30/100];
% Datos{6}.parent=Datos{1};

Folders=dir('f*.');
NumFolders=size(Folders,1);
FN=0;
FP=0;
TP=0;
TN=0;
FNaux=0;
FPaux=0;
TPaux=0;
TNaux=0;
FPR=0;
TPR=0;
ACC=0;
A=size(Datos,2);

for k=1:NumFolders
% for k=54:76
% k=24;
    cd(Folders(k).name);
    Images=dir('*.jpg');
    NumImages=size(Images,1);
    cd ..;
    MatrizPositions=labels(k,NumImages);

    %Para calcular la posicion de ambos ojos
    aux=MatrizPositions(:,5:12);
    aux1=aux(:,1:2:8)';
    aux2=aux(:,2:2:8)';
    rect=[min(aux1); min(aux2); max(aux1); max(aux2)]';
    MatrizPositions=[MatrizPositions rect];     %NumImagesx24(4
columnas para elemento de deteccion, cara, ojo izq, ojo dch,
    %nariz, boca, ambos ojos)

    for i=1:NumImages
%     i=14;
%         RectFeatureaux=[];
        RectFace=[];
%         RectFeature=[];
        Rectmixaux=[];
        Rectmix=[];
        refFeature=cell(1,A);
%         PositionFeatureaux=[];
        PositionFeature=cell(1,A);
%         refFeature2=[];
        refmix=[];
        index=[];

        tic;
        cd(Folders(k).name);
        img=imread(Images(i).name);
        cd ..;
```

```
        [Rectangle Rectanglemix RectangleFeature
r]=FaceDetection(img,Datos,MatrizPositions(i,1:4));

        q=0;
        p=0;
        d=0;

        figure(1);
        imshow(img);
        truesize;
        hold on;

        if r==1
            for s=1:A
                if size(Datos{1,s},1)==1
                    Position=MatrizPositions(i,1+q:4+q);

Rectangle2=[Position(:,1),Position(:,2);Position(:,3),Position(:,2);..
.

Position(:,3),Position(:,4);Position(:,1),Position(:,4);...
                        Position(:,1),Position(:,2)];

                    it=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(1,1);
                    ib=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(1,2);
                    il=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(1,3);
                    ir=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(1,4);
                    ot=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(2,1);
                    ob=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(2,2);
                    ol=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(2,3);
                    or=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(2,4);


rect1=[Position(:,1)+il,Position(:,2)+it,Position(:,3)-
ir,Position(:,4)-ib];
                    rect2=[Position(:,1)-ol,Position(:,2)-
ot,Position(:,3)+or,Position(:,4)+ob];

rectangle1=[rect1(:,1),rect1(:,2);rect1(:,3),rect1(:,2);rect1(:,3),rec
t1(:,4);rect1(:,1),rect1(:,4);rect1(:,1),rect1(:,2)];

rectangle2=[rect2(:,1),rect2(:,2);rect2(:,3),rect2(:,2);rect2(:,3),rec
t2(:,4);rect2(:,1),rect2(:,4);rect2(:,1),rect2(:,2)];

%                     plot(0,'g');
%                     plot (Rectangle2(:,1), Rectangle2(:,2), 'r');
%                     plot (rectangle1(:,1), rectangle1(:,2), 'b--');
%                     plot (rectangle2(:,1), rectangle2(:,2), 'b--');
                end
                p=p+2;
                q=q+4;
            end
```

```
            plot (Rectangle(:,1), Rectangle(:,2), 'g');

%Cálculo TP,FP,TN,FN

            refRectangle=[];
            for c=1:2:size(Rectangle,2)

refRectangleaux=[Rectangle(1,c:c+1),Rectangle(3,c:c+1)];
                refRectangle=[refRectangle;refRectangleaux];
            end

            if isnan(Rectangle(:,:))==0
                for w=1:size(refRectangle,1)
                    if refRectangle(w,1:2)>=rect2(1,1:2) &
refRectangle(w,3:4)<=rect2(1,3:4)&...
                            refRectangle(w,1:2)<=rect1(1,1:2) &
refRectangle(w,3:4)>=rect1(1,3:4)
                        TP=TP+1;
                    else
                        FP=FP+1;
                    end
                end
            else
                if isnan(Position(:,:))==0
                    FN=FN+1;
                else
                    TN=TN+1;
                end
            end

        else   %detector mixto(cara y caracteristicas faciales)

            for s=1:A
                if size(Datos{1,s},1)==1
                    Position=MatrizPositions(i,1+q:4+q);

Rectangle2=[Position(:,1),Position(:,2);Position(:,3),Position(:,2);..
.

Position(:,3),Position(:,4);Position(:,1),Position(:,4);...
                        Position(:,1),Position(:,2)];

                    it=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(1,1);
                    ib=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(1,2);
                    il=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(1,3);
                    ir=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(1,4);
                    ot=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(2,1);
                    ob=(Position(:,3)-Position(:,1))*
Datos{1,s}.percents(2,2);
                    ol=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(2,3);
                    or=(Position(:,4)-Position(:,2))*
Datos{1,s}.percents(2,4);
```

```
rect1=[Position(:,1)+il,Position(:,2)+it,Position(:,3)-
ir,Position(:,4)-ib];
                    rect2=[Position(:,1)-ol,Position(:,2)-
ot,Position(:,3)+or,Position(:,4)+ob];

rectangle1=[rect1(:,1),rect1(:,2);rect1(:,3),rect1(:,2);rect1(:,3),...

rect1(:,4);rect1(:,1),rect1(:,4);rect1(:,1),rect1(:,2)];

rectangle2=[rect2(:,1),rect2(:,2);rect2(:,3),rect2(:,2);rect2(:,3),...

rect2(:,4);rect2(:,1),rect2(:,4);rect2(:,1),rect2(:,2)];

%                     plot(0,'g');
%                     plot (Rectangle2(:,1), Rectangle2(:,2), 'r');
%                     plot (rectangle1(:,1), rectangle1(:,2), 'b--');
%                     plot (rectangle2(:,1), rectangle2(:,2), 'b--');

                    for c=1:2:size(RectangleFeature{1,s},2)

refFeatureaux=[RectangleFeature{1,s}(1,c:c+1),RectangleFeature{1,s}(3,
c:c+1)];

refFeature{1,s}=[refFeature{1,s};refFeatureaux];
                    end

                    if s~=1
                        PositionFeatureaux=[rect1;rect2];
                        PositionFeature{1,s}=[PositionFeature{1,s}
PositionFeatureaux];
                    else
                        PositionFace=[rect1;rect2];
                    end

                end
                p=p+2;
                q=q+4;
            end

            for c=1:2:size(Rectanglemix,2)

refmixaux=[Rectanglemix(1,c:c+1),Rectanglemix(3,c:c+1)];
                refmix=[refmix;refmixaux];
            end

            %Seleccion de candidatos a cara. El candidato debe
contener al menos
            %tantas caracteristicas en su interior como
caracteristicas
            %a detectar

            for s=1:size(refFeature,2)
                indexaux=isempty(refFeature{1,s});
                index=[index indexaux];
            end
            d=find(index==min(index));

            if size(d,2)==r-1
                for j=1:size(Rectanglemix,2)/2
                    loop=0;
```

```
                    for s=1:size(d,2)
                        for b=1:size(refFeature{1,d(:,s)},1)
                            if
refFeature{1,d(:,s)}(b,1:2)>=refmix(j,1:2) & ...

refFeature{1,d(:,s)}(b,3:4)<=refmix(j,3:4)
                                loop=loop+1;
                            end
                        end
                    end
                    if loop>=r-1
                            Rectmixaux=[refmix(j,1:2);refmix(j,3)
refmix(j,2);...
                                 refmix(j,3:4);refmix(j,1)
refmix(j,4);...
                                    refmix(j,1:2)];

                    end
                    Rectmix=[Rectmix,Rectmixaux];
                    Rectmixaux=[];
                end
            else
                Rectmix=[];
            end

                refmix2=[];
                for c=1:2:size(Rectmix,2)
                    refmixaux2=[Rectmix(1,c:c+1),Rectmix(3,c:c+1)];
                    refmix2=[refmix2;refmixaux2];
                end

                if size(refmix2,1)>1
                    for w=1:size(refmix2,1)-1
                        if refmix2(w,1:2)>=refmix2(w+1,1:2) &
refmix2(w,3:4)<=refmix2(w+1,3:4)|...
                            refmix2(w+1,1:2)>=refmix2(w,1:2) &
refmix2(w+1,3:4)<=refmix2(w,3:4)

                              refmix3=[refmix2(w,1:2) refmix2(w,3:4)];
                        else
                            refmix3=refmix2;
                        end
                    end
                else
                   refmix3=refmix2;
                end

            %Cálculo TP,FP,TN,FN

            cont=0;
            if size(Rectmix,2)~=0
                for w=1:size(refmix3,1)
                    for s=1:size(d,2)
                        for b=1:size(refFeature{1,d(:,s)},1)
                            if
refFeature{1,d(:,s)}(b,1:2)>=PositionFeature{1,d(:,s)}(2,1:2) & ...

refFeature{1,d(:,s)}(b,3:4)<=PositionFeature{1,d(:,s)}(2,3:4)&...

refFeature{1,d(:,s)}(b,1:2)<=PositionFeature{1,d(:,s)}(1,1:2)&...
```

```
refFeature{1,d(:,s)}(b,3:4)>=PositionFeature{1,d(:,s)}(1,3:4)
                                TPaux=TPaux+1;
                                cont=cont+1;
                            else
                                FPaux=FPaux+1;
                            end

                        end
                    end
                end
            else
                if isnan(Position(:,:))==0
                    FNaux=FNaux+1;
                    FN=FN+1;
                else
                    TNaux=TNaux+1;
                    TN=TN+1;
                end
            end

            if FNaux==0 & TNaux==0
                if cont>=r-1
                    TP=TP+1;
                     RectFace=[refmix3(w,1:2);refmix3(w,3)
refmix3(w,2);...
                              refmix3(w,3:4);refmix3(w,1)
refmix3(w,4);...
                                  refmix3(w,1:2)];
                else
                    if FPaux~=0
                        FP=FP+1;
                    else
                        TP=TP+1;
                    end
                end
            end

            FNaux=0;
            TNaux=0;

            for g=1:2:size(RectFace,2)
                plot(RectFace(:,g), RectFace(:,g+1), 'g');
            end
        end
%         pause

%         hold off;

        FPR=FP/(FP+TN);
        TPR=TP/(TP+FN);
        SPC=1-FPR;
        ACC=((TP+TN)/(FP+TN+TP+FN))*100;
        time=toc;

        if size(img,2)>1200
            text(-105,-5,strcat('IPS=',num2str(round(1/time))));...
                text(-105,15,strcat('ACC=',num2str(ACC)));
            text(-105,35,strcat('FP=',num2str(FP)));...
                text(-105,55,strcat('FN=',num2str(FN)));...
                text(-105,75,strcat('TP=',num2str(TP)));
```

```
            text(-105,95,strcat('TN=',num2str(TN)));...
                text(-105,115,strcat('TPR=',num2str(TPR)));...
                text(-105,135,strcat('SPC=',num2str(SPC)));
            text(-105,155,strcat('FPR=',num2str(FPR)));
        else
            text(-75,-5,strcat('IPS=',num2str(round(1/time))));...
                text(-75,15,strcat('ACC=',num2str(ACC)));
            text(-75,35,strcat('FP=',num2str(FP)));...
                text(-75,55,strcat('FN=',num2str(FN)));...
                text(-75,75,strcat('TP=',num2str(TP)));
            text(-75,95,strcat('TN=',num2str(TN)));...
                text(-75,115,strcat('TPR=',num2str(TPR)));...
                text(-75,135,strcat('SPC=',num2str(SPC)));
            text(-75,155,strcat('FPR=',num2str(FPR)));
        end

%         h=legend('localización detectada','localización
real','límites',4);
%         set(h,'EdgeColor',[1,1,1]);
    end
end

cont=0;
AUX=[];
for w=1:A
    if size(Datos{1,w},1)==1
        titulo=strrep(Datos{w}.xml,'_','-');
        cont=cont+1;
        Array{cont,1}=titulo;
        AUX=[AUX,titulo];
    end
end

%Tabla
f = figure('Position',[70 650 660 135]);
dat = {ACC,FP,FN,TP,TN,TPR,SPC,FPR};
cnames = {' ACC (%) ',' FP ',' FN ',' TP ',' TN ',' TPR ',' SPC(TNR)
',' FPR '};
t =
uitable('BackgroundColor',[0.7,0.5,1],'Data',dat,'ColumnName',cnames,'
Parent',f,...
    'Position',[17 2 700 35]);
text(0.35,0.66,Array);      %title
set(gca,'Visible','off');

%Guardar tabla
set(gcf,'PaperpositionMode','auto');
y=strrep(AUX,'.xml','__');
saveas(f,strrep(y,'haarcascade-',[]),'jpg');

clear all
% close all
```

8.2 ANEXO 2. FUNCIÓN EN C++

El programa de detección está implementado en lenguaje C++, por lo que para poder utilizarlo desde matlab, hay que realizar una compilación previa de éste, es decir, establecer una ruta de acceso en matlab, siguiendo los siguientes pasos:

- Configuración Mex del compilador:
  "*mex-setup*" en la ventana de comandos de MATLAB. Siguiendo las instrucciones se elige el compilador apropiado, en nuestro caso Visual Studio 2008.
- Cambio de ruta de acceso al / src / y ejecutar el comando mex FaceDetect.cpp -I../Include / .. / lib / *. lib-outdir .. / bin / en la ventana de comandos. Los archivos compilados se almacenan en el directorio bin. Hay que colocar estos archivos de salida junto con "cv200.dll" y "cxcore.dll" en el directorio deseado y la ruta de valor apropiado en Matlab.

8.2.1 cvod200uint_v02.cpp

Entradas:

```
(<HaarCascade File>, <GrayImage>, <Scale>,<DebugLevel>,<minsize>)
```

- HaarCascade File
  Archivo xml.

- GrayImage
  Imagen a detectar en escala de grises.

- Scale
  Escala.

- DebugLevel
  Nivel de depuración.

- minsize
  Tamaño mínimo de la ventana donde se realizan las búsquedas.

```
/*********************************************************************
***
 *

*********************************************************************
***/

#include "mex.h" // Required for the use of MEX files

#include "cv.h" // Required for OpenCV

#include "highgui.h" // Required for Debug
#define HAS_OPENCV

#include "mc_convert.h"
```

```
#include <algorithm>
#include <iterator>
#include <iostream>
using namespace std;

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;

/** This is a the only prototype function that you need to get a mex
file to work. */
void mexFunction(int output_size, mxArray *output[], int input_size,
const mxArray *input[])
{
    char *input_buf;
    int buflen;
    mxArray *xData;
    unsigned char *xValues;
    int i,j;
    double scale,debuglevel,minsize;
    int NoOfCols, NoOfRows;

    /* check for proper number of arguments */
    if(input_size!=5)
      mexErrMsgTxt("Usage: ObjectDetect (<HaarCascade File>,
<GrayImage>, <Scale>,<DebugLevel>,<minsize>)");

    /* input must be a string */
    if ( mxIsChar(input[0]) != 1)
      mexErrMsgTxt("Input 1 must be a string.");

    /* get the length of the input string */
    buflen = (mxGetM(input[0]) * mxGetN(input[0])) + 1;

    /* copy the string data from input[0] into a C string input_ buf.
*/
    input_buf = mxArrayToString(input[0]);

    if(input_buf == NULL)
      mexErrMsgTxt("Could not read HarrCascade Filename to string.");

    // Read the Haar Cascade from the XML file
    cascade = (CvHaarClassifierCascade*) cvLoad( input_buf, 0, 0, 0);
    if( !cascade )
    {
       mexErrMsgTxt("ERROR: Could not load classifier cascade" );
    }

    /* Check if the input image is in uint8 format*/
    if (!(mxIsUint8(input[1]))) {
      mexErrMsgTxt("Input image must be gray scale and of type
uint8");
    }

    //Copy input pointer
    // This carries the input grayscale image that was sent from
Matlab
    xData = (mxArray *)input[1];

    //Get the matrix from the input data
    // The matrix is rasterized in a column wise read
    //xValues =  (UINT8_T *) mxGetData(xData);
```

```
      //xValues =  mxGetPr(xData);

    NoOfCols = mxGetN(xData); // Gives the number of Columns in the
image
    NoOfRows = mxGetM(xData); // Gives the number of Rows in the image

    /* Get the number of dimensions in the input Image */
    int number_of_dims = mxGetNumberOfDimensions(input[1]);
    if (number_of_dims > 2)
        mexErrMsgTxt("Input image should be gray scale and of type
uint8");

    /* Check if the scale is double*/
    if (!(mxIsDouble(input[2]))) {
      mexErrMsgTxt("Scale must be a number of type double");
    }
    scale=mxGetScalar(input[2]);

      /* Check if the debuglevel is int*/
    if (!(mxIsDouble(input[3]))) {
      mexErrMsgTxt("DebugLevel must be a number of type double");
    }
    debuglevel=mxGetScalar(input[3]);

      /* Check if the minsize is int*/
    if (!(mxIsDouble(input[4]))) {
      mexErrMsgTxt("DebugLevel must be a number of type double");
    }
    minsize=mxGetScalar(input[4]);

    // Create an IplImage from the data so face detection can be run
on it
    //IplImage* gray = cvCreateImage( cvSize(NoOfCols, NoOfRows),
IPL_DEPTH_8U, 1 );
      IplImage* gray = mxArr_to_new_IplImage (xData);
      IplImage* smallImg = cvCreateImage(
cvSize(cvRound(NoOfCols/scale), cvRound(NoOfRows/scale)),
IPL_DEPTH_8U, 1 );

    // Load the column wise vector into the IplImage
    // IplImage data is read in a rowwise manner
    // Appropriate conversion is carried out here
    /*for(i=0;i<NoOfCols;i++)
       for(j=0;j<NoOfRows;j++)
        {
            int value = (int) xValues[(i*NoOfRows)+j];
            gray->imageData[j*NoOfCols+i] = value;
        }*/



/*********************************************************************
     *                                Detect objects
*

*********************************************************************/

    cvResize( gray, smallImg, CV_INTER_LINEAR );
    //cvPyrDown(gray,smallImg,CV_GAUSSIAN_5x5);
      if (debuglevel>=1){
            cv::imshow("result", gray);
```

```
      }

    //cvEqualizeHist( smallImg, smallImg );

    storage = cvCreateMemStorage(0);

    CvSeq* objects = cvHaarDetectObjects( smallImg, cascade, storage,
                                          1.1, 2,
0/*|CV_HAAR_SCALE_IMAGE*/|CV_HAAR_DO_CANNY_PRUNING,
                                          cvSize(minsize, minsize)
);


    // Allocate output variable
    // Number of rows = number of detected faces
    // Number of columns = 4
    // 1: Location X of the face
    // 2: Location Y of the face
    // 3: Width of the face
    // 4: Height of the face
    double *Data;
    if (objects->total)
    {
        output[0] = mxCreateDoubleMatrix(objects->total, 4, mxREAL);
        Data = mxGetPr(output[0]); // Get the pointer to output
variable
        // Iterate trou each of the detected faces
        for( i = 0; i < objects->total; i++ )
        {
            CvRect* r = (CvRect*)cvGetSeqElem( objects, i );
           /* The Data pointer again has to be filled in a column wise
manner
            * The first column will contain the x location of all
faces
            * while column two will contain y location of all faces */
           Data [i] = scale*r->x;
           Data [i+objects->total] = scale*r->y;
           Data [i+objects->total*2] = scale*r->width;
           Data [i+objects->total*3] = scale*r->height;
           // Debug
           // printf ("%d %d %d %d\n", r->x, r->y, r->width, r-
>height);
        }
    }
    else
    {
        output[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
        Data = mxGetPr(output[0]); // Get the pointer to output
variable
        Data[0] = -1;
    }

    // Release all the memory
    cvReleaseImage( &gray );
      cvReleaseImage( &smallImg );
    cvReleaseMemStorage( &storage );
    cvReleaseHaarClassifierCascade( &cascade );
      mxFree(input_buf);
    return;
}
```

8.3 ANEXO 3. Función de detección de objetos de openCV

A continuación se muestra la función principal de detección de objetos de OpenCV; cvHaarDetectObjects. Esta función, aplicada en este trabajo, depende de los parámetros image, cascade, storage, storagescale_factor, min_neighbors, flags, min_size.

8.3.1 cvHaarDetectObjects

Detecta objetos en la imagen

typedef struct CvAvgComp
{
 CvRect rect; / * rectángulo delimitador de la cara (rectángulo promedio de un grupo) *
/
 int neighbors; / * número de rectángulos vecinos en el grupo * /
}
CvAvgComp;

CvSeq* cvHaarDetectObjects (image, cascade, storage, scale_factor, min_neighbors, flags, min_size);

```
CvSeq* cvHaarDetectObjects( const IplImage* img,
CvHidHaarClassifierCascade* cascade,
 CvMemStorage* storage, double scale_factor=1.1,
 int min_neighbors=3, int flags=0, cvSize );
```

- Image
  Imagen para detectar objetos de entrada.

- Cascade
  Clasificador Haar en cascada en la representación interna.

- Storage
  Memoria de almacenamiento para almacenar la secuencia resultante de los rectángulos candidatos a objeto.

- Scale_factor
  El factor por el que se escala la ventana de búsqueda entre los siguientes ciclos, por ejemplo, 1.1 significa aumentar la ventana en un 10%.

- min_neighbors
  Número mínimo (menos uno) de los rectángulos vecinos que constituyen un objeto. Todos los grupos con un menor número de rectángulos que min_neighbors-1 son rechazados. Si min_neighbors es 0, la función no devuelve ningún grupo en absoluto, devuelve todos los rectángulos detectados como candidatos, que puede ser útil si el usuario desea aplicar un procedimiento de agrupación personalizada.

- Flags
  Modo de funcionamiento. Hay varios flags:
  //|CV_HAAR_FIND_BIGGEST_OBJECT,
  //|CV_HAAR_DO_ROUGH_SEARCH,

|CV_HAAR_DO_CANNY_PRUNING y
//|CV_HAAR_SCALE_IMAGE.
Si se establece el flag CV_HAAR_DO_CANNY_PRUNING, la función puede utilizar el detector de bordes Canny para rechazar algunas regiones de la imagen que tienen muy pocos o demasiados bordes y por lo tanto no puede contener el objeto buscado. Los valores umbral particulares están afinados para la detección de la cara y en este caso la exclusión acelera el proceso (cuando las regiones son planas se evita procesarlas). CV_HAAR_FIND_BIGGEST_OBJECT limita la detección a la cara más grande.

- min_size
  Es el tamaño de la ventana más pequeño en el cual se hace la búsqueda, colocando este a un mayor valor se reducirá el proceso de computación evitando encontrar rostros pequeños.

La función cvHaarDetectObjects encuentra regiones rectangulares en la imagen que pueden contener los objetos buscados, la cascada ha sido entrenada para esas regiones, y los devuelve como una secuencia de rectángulos. La función analiza la imagen varias veces a diferentes escalas (ver cvSetImagesForHaarClassifierCascade). Cada vez que considere regiones superpuestas en la imagen se aplican los clasificadores a las regiones con cvRunHaarClassifierCascade. También puede aplicar un poco razonamiento heurístico para reducir el número de regiones analizadas, tales como poda Canny. Después del procesamiento y la reagrupación de los rectángulos candidatos (regiones que pasan la cascada del clasificador), se agrupan y devuelven como una secuencia de rectángulos promedio de cada grupo lo suficientemente grande.

8.3.2 cvSetImagesForHaarClassifierCascade

Asigna las imágenes a la cascada oculta

```
void cvSetImagesForHaarClassifierCascade( CvHidHaarClassifierCascade* cascade,
 const CvArr* sumImage, const CvArr* sqSumImage,
 const CvArr* tiltedImage, double scale );
```

- Cascade
  Clasificador en cascada oculto, creado por cvCreateHidHaarClassifierCascade.

- sumImage
  Integral (suma) de una imagen en formato entero de 32 bits. Esta imagen, así como las dos imágenes siguientes se utilizan para la evaluación de función rápida y brillo / contraste de la normalización. Todos ellos pueden ser recuperados a partir de la imagen de entrada por un canal de 8 bits utilizando la función cvIntegral. Tenga en cuenta que todas las imágenes son de 1 pixel más anchas y de un pixel más altas que la imagen original de 8 bits.

- sqSumImage
  Cuadrado de la suma imagen en un formato de 64 bits.
- tiltedSumImage
  Pendiente de la suma imagen en formato entero de 32 bits.

- scale
  Ventana de escala de la cascada. Si la escala = 1, se utiliza el tamaño de la ventana original (se buscan objetos de ese tamaño), si escala = 2, se emplea una ventana dos veces más grande. Mientras esto acelera la búsqueda

alrededor de cuatro veces, caras de tamaño inferior a 48x48 pueden no ser detectadas.

La función cvSetImagesForHaarClassifierCascade asigna imágenes y/o ventanas de escala al clasificador de cascada oculta. La función se utiliza para preparar la cascada para la detección de objetos del tamaño concreto en la imagen concreta. La función se llama internamente por cvHaarDetectObjects, pero puede ser llamado por el usuario si es necesario en el uso de cvRunHaarClassifierCascade.

8.3.3 cvRunHaarClassifierCascade

Se ejecuta la cascada boosted del clasificador dada la ubicación en la imagen.

int cvRunHaarClassifierCascade(CvHidHaarClassifierCascade* cascade,
CvPoint pt, int startStage=0);

- cascada
  Clasificador Haar en cascada oculto.

- pt
  Ángulo superior izquierdo de la región analizada. El tamaño de la región es el tamaño de la ventana original a escala con la escala establecida actualmente. El tamaño de la ventana actual se puede recuperar con la función cvGetHaarClassifierCascadeWindowSize.

- startStage
  Índice inicial de base cero de la etapa inicial de la cascada. La función asume que todas las etapas anteriores fueron superadas. Esta función se usa internamente por cvHaarDetectObjects para una mejor utilización de la caché del procesador.

La función cvRunHaarHaarClassifierCascade ejecuta un clasificador Haar en cascada en una única localización de la imagen. Antes de utilizar esta función las imágenes integral y la escala apropiada (tamaño de la ventana) deben establecerse mediante cvSetImagesForHaarClassifierCascade. La función devuelve un valor positivo si los rectángulos analizados (candidatos) han superado todas las etapas del clasificador y devuelve cero o un valor negativo si no las superan.

8.4 ANEXO 4. ELEMENTOS COMPUTACIONALES

A continuación se procede a detallar las herramientas más importantes que han sido utilizadas para este proyecto, incluyendo aplicaciones software, compiladores, sistemas operativos y aplicaciones para experimentación y estudio de conceptos generales.

8.4.1 MEX Files

MEX Files se refiere a Matlab Executable Files (archivos ejecutables Matlab). Estos archivos son subrutinas enlazadas de manera dinámica desde código fuente C, C++ o Fortran, el cual una vez compilado puede lanzarse desde Matlab del mismo modo que los archivos .m. Gracias a las funciones de interfaces externas se obtiene la funcionalidad para transferir datos entre los archivos MEX y Matlab, y la habilidad para

llamar a las funciones Matlab desde C/C++. Los componentes de un archivo MEX de C/C++ son los siguientes:

- Rutina gateway que sirve de interfaz entre C/C++ y Matlab. Es el punto de entrada a la librería compartida del archivo MEX. A través de esta rutina Matlab accede al resto de rutinas de los archivos MEX. El nombre de la rutina ha de ser: mexFunction. Un ejemplo sería:

```
void mexFunction(
 int nlhs, mxArray *plhs[],
 int nrhs, const mxArray *prhs[])
{
 /* more C/C++ code ... */
}
```

  Este tipo de rutinas debe contener los siguientes parámetros:
  1. prhs: Array de argumentos de entrada.
  2. plhs: Array de argumentos de salida.
  3. nrhs: El número de argumentos de entrada, es decir, el tamaño del array prhs.
  4. nlhs: El número de argumentos de salida, es decir, el tamaño del array plhs.

- Rutina computacional, que es la encargada de realizar las operaciones para las que se construye el archivo.

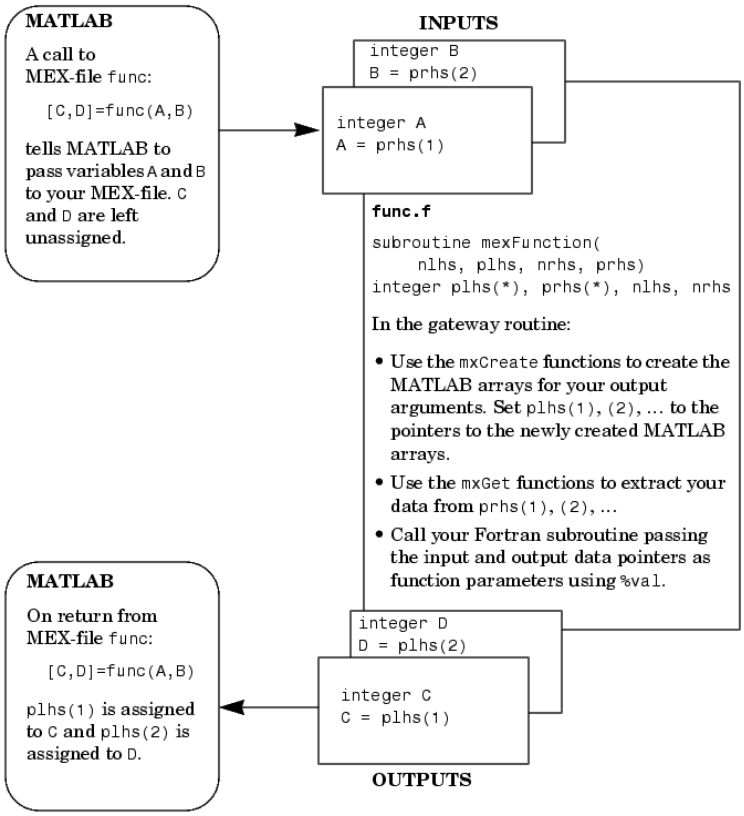


Figura. 8.1. Flujo de datos en archivos MEX

En este ejemplo la sintaxis de la función representada es `[C, D] = func(A,B).` La rutina gateway en `func.c` utiliza la función `mxCreate*` para crear los arrays de Matlab para las variables de salida y relaciona los punteros `plhs[0]` y `plhs[1]` con

estas variables. Utiliza las funciones `mxGet*` para extraer los datos de entrada que se le envían a las funciones desde Matlab. Por último se llama a las rutinas computacionales para realizar las operaciones necesarias.

8.4.2 XML

XML (Extensible Markup Language, lenguaje de marcado extensible) es una forma de reutilizar datos en un archivo o de automatizar el proceso de sustitución de datos en un archivo con datos de otro archivo. Es decir, XML es un estándar para el intercambio de información estructurada entre diferentes plataformas. XML utiliza etiquetas para describir partes de un archivo, por ejemplo, un encabezado o un artículo. Estas etiquetas marcan los datos de manera que se pueden almacenar en un archivo XML y se puedan gestionar adecuadamente cuando se exporten a otros archivos. XML es como un mecanismo de conversión de datos. XML etiqueta el texto de la etiqueta y otro contenido en un archivo de forma que las aplicaciones puedan reconocer y presentar los datos.

XML se considera un lenguaje extensible porque cada uno crea sus propias etiquetas XML. Se puede crear una etiqueta para cada tipo de información que se desee reutilizar. Las etiquetas XML no contienen información acerca de cómo se deben mostrar los datos ni sobre el formato que deben tener. Las etiquetas XML sirven estrictamente para identificar contenido.

8.4.3 OpenCV

OpenCV es un conjunto de librerías de código libre de visión artificial originalmente creada por Intel y disponible en http://SourceForge.net/projects/opencvlibrary. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella ex-presadas. La librería está escrita en C y C++ y funciona en Linux, Windows y Mac OS X. OpenCV fue diseñado para conseguir una gran eficiencia computacional concentrándose activamente sobre las aplicaciones en tiempo real. Su librería posee más de 500 algoritmos optimizados (y se utiliza ampliamente alrededor del mundo. Es un sistema muy relevante y ampliamente utilizado ya que tiene más de 2 millones de descargas y más de 40000 personas en su grupo de usuarios.

La importancia de estas librerías se debe a que han sido ampliamente evaluadas y mejoradas por un amplio grupo de personas con lo que se ha obtenido un sistema sólido. De entre todas las funcionalidades que tiene, las que son utilizadas en este proyecto son las que se relacionan con la detección facial y la detección de características faciales, que han sido extraídas de la versión 2.0 de OpenCV.

Los ficheros necesarios para la implementación de la detección facial son: haarcascade_frontalface_default.xml, haarcascade_frontalface_alt2.xml, haarcascade_frontalface_alt.xml y haarcascade_frontalface_alt_tree.xml. Los necesarios para la detección de características faciales son:

haarcascade_mcs_righteye.xml, haarcascade_righteye_2splits.xml
haarcascade_lefteye_2splits.xml, haarcascade_mcs_lefteye.xml

haarcascade_mcs_nose.xml haarcascade_mcs_mouth.xml haarcascade_mcs_eyepair_big.xml, haarcascade_mcs_eyepair_small.xml.

- haarcascade_frontalface_default.xml: La cascada fue desarrollada por Rainer Lienhart utilizando el algoritmo Adaboost con imágenes de 24x24 y obteniendo 25 etapas en el detector.

- haarcascade_frontalface_alt2.xml: Archivo implementado por Rainer Lienhart basándose en el algoritmo Gentle Adaboost, el cual minimiza la función exponencial de pérdidas de Adaboost mediante aproximaciones utilizando el método de Newton. Sus imágenes de entrenamiento son de 20x20 píxeles y las etapas utilizadas son 20.

- haarcascade_frontalface_alt.xml: Basado en el algoritmo Gentle Adaboost utilizando imágenes de entrenamiento de 20x20 pixeles. Creado por Rainer Lienhart.

- Haarcascade_frontalface_alt_tree.xml: Fundamentado en el algoritmo Gentle Adaboost utilizando imágenes de 20x20 pixeles. Este detector utiliza estados en forma de árbol en vez de clasificadores en cascada. Creado por Rainer Lienhart.

- haarcascade_mcs_righteye.xml: 18x12 Right eye (in the image) detector computed with 7000 positive samples.

- haarcascade_righteye_2splits.xml: Tree-based 20x20 right eye detector. The detector is trained by 6665 positive samples from FERET, VALID and BioID face databases. Creado por Shiqi Yu.

- haarcascade_lefteye_2splits.xml: Tree-based 20x20 left eye detector. The detector is trained by 6665 positive samples from FERET, VALID and BioID face databases. Creado por Shiqi Yu.

- haarcascade_mcs_lefteye.xml: 18x12 Left eye (in the image) detector computed with 7000 positive samples.

- haarcascade_mcs_nose.xml: 18x15 Nose detector computed with 7000 positive samples.

- haarcascade_mcs_mouth.xml: 25x15 Mouth detector computed with 7000 positive samples.

- haarcascade_mcs_eyepair_big.xml: 45x11 Eye pair detector computed with 7000 positive samples.

- haarcascade_mcs_eyepair_small.xml: 22x5 Eye pair detector computed with 7000 positive samples.

8.4.4 Matlab

Matlab (MATrix LABoratory) es un programa interactivo para computación numérica. Este programa es muy utilizado por ingenieros en el análisis y diseño, ya que posee una gran versatilidad y capacidad para resolver problemas en matemáticas, física, química, ingeniería, finanzas... mediante ejecución interactiva de comandos o

mediante la escritura de programas sencillos en un lenguaje de programación propio (ficheros con extensión .m). Además, tiene gran aceptación en escuelas, centros universitarios, departamentos de investigación, etc.

El elemento principal de la aplicación son las matrices (de ahí el nombre del programa), con las cuales es posible realizar operaciones en menos tiempo que con otros lenguajes. Su punto débil es que es la ejecución es interpretada, lo cual repercute muy negativamente en la velocidad general, sobre todo en los bucles. Para solucionar este problema, Matlab permite ejecutar librerías dinámicas escritas en C/C++ que pueden ser compiladas con el comando *mex*.

En el presente trabajo, Matlab se utiliza como herramienta central sobra la cual se implementan y se prueban los algoritmos.

8.4.5 Windows XP

Windows XP (cuyo nombre en clave inicial fue *Whistler*) es una versión de Microsoft Windows, línea de sistemas operativos desarrollado por Microsoft. Lanzado al mercado el 25 de octubre de 2001, actualmente es el sistema operativo para x86 más utilizado del planeta (con una cuota de mercado del 48,9%) y se considera que existen más de 400 millones de copias funcionando. Las letras "XP" provienen de la palabra *eXPeriencia* (*eXPerience* en inglés).

Dispone de versiones para varios entornos informáticos, incluyendo PCs domésticos o de negocios, equipos portátiles, "netbooks", "tablet PC" y "media center". Sucesor de Windows 2000 junto con Windows ME, y antecesor de Windows Vista, es el primer sistema operativo de Microsoft orientado al consumidor que se construye con un núcleo y arquitectura de Windows NT disponible en versiones para plataformas de 32 y 64 bits.

A diferencia de versiones anteriores de Windows, al estar basado en la arquitectura de Windows NT proveniente del código de Windows 2000, presenta mejoras en la estabilidad y el rendimiento. Tiene una interfaz gráfica de usuario (GUI) perceptiblemente reajustada (denominada *Luna*), la cual incluye características rediseñadas, algunas de las cuales se asemejan ligeramente a otras GUI de otros sistemas operativos, cambio promovido para un uso más fácil que en las versiones anteriores. Se introdujeron nuevas capacidades de gestión de software para evitar el "*DLL Hell*" (*infierno de las DLLs*) que plagó las viejas versiones. Es también la primera versión de Windows que utiliza la activación del producto para reducir la piratería del software, una restricción que no sentó bien a algunos usuarios. Ha sido también criticado por las vulnerabilidades de seguridad, integración de Internet Explorer, la inclusión del reproductor Windows Media Player y aspectos de su interfaz.

8.4.6 Compilador: Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Microsoft Visual Studio 97 fue la primera versión que salió al mercado, ésta incluía Visual Basic y Visual C++ 5.0 para realizar software para Windows específicamente, mientras que Visual J++ era para Java y Windows.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión net 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Microsoft Visual Studio .NET 2002 fue un salto completamente drástico, ya que cambia completamente la estructura. Ahora con .NET los programas no se compilan para generar un ejecutable máquina (un archivo .EXE conocido comúnmente para el ambiente Microsoft Windows), sino que genera un archivo intermedio para ser ejecutado en diferentes plataformas (distintas arquitecturas de software y hardware, como GNU/Linux, Solaris de Sun Microsystems o Mac OS X de Apple).

8.5 ANEXO 5. ENTRENAMIENTO USANDO HAARTRAINING.

La librería OpenCV, provee programas o comandos los cuales son usados para entrenar clasificadores llamado HaarTraining por lo tanto es posible crear clasificadores propios haciendo uso de éstas. Estos comandos son: “createsamples.exe”, “haartraining.exe” y “performance.exe”. Por otro lado, OpenCV viene con unos clasificadores ya entrenados para detección facial frontal y otros objetos en archivos con extensión xml. El objeto de interés puede ser cualquier objeto como árbol, fruta, coche, etc., no solamente el rostro. Para conseguir este clasificador es preciso completar unos ciertos pasos que quedan descritos a continuación.

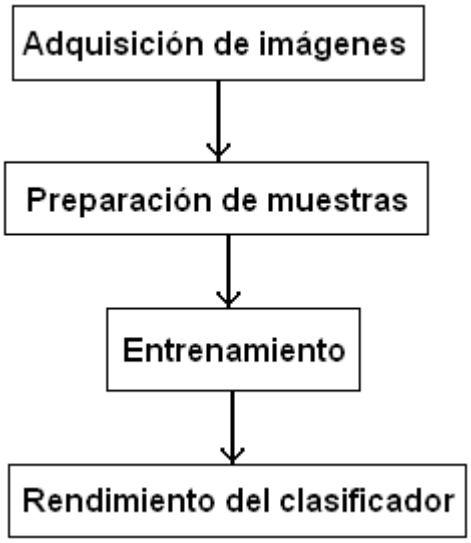


Figure 8.2. Etapas de entrenamiento usando Haartraining.

8.5.1 Adquisición de imágenes

La parte más importante del proceso de entrenamiento del clasificador es la de la recolección de una gran cantidad de muestras positivas y negativas, de forma que de ellas se puedan extraer las mejores reglas posibles que formen un detector de objetos potente. El conjunto de imágenes se divide en dos grupos; muestras positivas y muestras negativas.

Negativas: imagenes en las cuales no aparece el objeto sobre el cual queremos generar el entrenamiento.

Positivas: imagenes que contienen el objeto que estamos intentando detectar. La idea es generar el objeto en distintos ángulos, iluminación, etc. En este archivo se especifica donde se encuentra el objeto dentro de la imagen positiva.

8.5.2 Preparación de muestras

En esta etapa nos valdremos de una herramienta llamada createsamples para la construcción de las muestras positivas necesarias. Tras su ejecución se crea un archivo de extensión vec que sirve al clasificador para su entrenamiento. La llamada al programa para crear el archivo de extensión vec tiene los siguientes argumentos:

-info: ubicación del archivo con el índice de las imagenes positivas.

-vec: nombre del archivo de salida con la muestra generada.

-num: cantidad de imagenes positivas.

-w: ancho de la muestra de salida.

-h: alto de la muestra de salida.

8.5.3 Entrenamiento

Para esta etapa nos valdremos de otra herramienta llamada haartraining, el archivo índice generado con las imagenes negativas y el archivo de muestras generado en la etapa anterior.

Nuevamente desde la línea de comandos:

-data: directorio de salida para la generación del entrenamiento, donde se almacena el clasificador.

-vec: archivo con muestras generado en la etapa anterior.

-bg: archivo índice con las imagenes negativas.

-nstages: numero de etapas de entrenamiento. Es directamente proporcional a la calidad del clasificador a generar.

-nsplit: debilidad del clasificador.

-minhitrate: minima tasa de detección en cada etapa.

-maxfalsealarm: determina el rango máximo de falsas alarmas admitidas para cada etapa.

-npos: numero de imagenes positivas indexadas.

-nneg: numero de imagenes negativas indexadas.

-w: ancho.

-h: alto.

-mem: memoria a utilizar en el proceso.

8.5.4 Rendimiento del clasificador

Una vez ejecutado el programa Haartaining, el clasificador puede ser probado con el comando Performance. Es conveniente realizar el test del clasificador con muestras distintas a las utilizadas durante el entrenamiento.

El comando performance tiene los siguientes argumentos:

- data: directorio donde se almacena el clasificador.

-info: ubicación del archivo con el índice de las imagenes positivas.

- maxSizeDiff y –maxPossDiff: determinan la coincidencia de detección de los rectángulos.

- sf: parámetro de detección.

-w: ancho de la muestra de salida.

-h: alto de la muestra de salida.